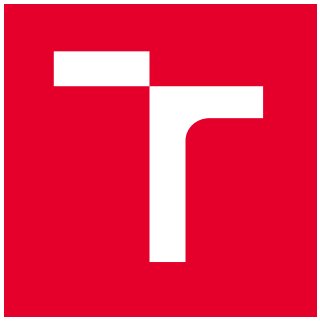


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

SIMULATION OF ROBOTIC SEARCH OF LOST RADIATION SOURCES

SIMULACE ROBOTICKÉHO VYHLEDÁVÁNÍ ZTRACENÝCH RADIAČNÍCH ZDROJŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. Miloš Cihlář

SUPERVISOR
VEDOUCÍ PRÁCE

prof. Ing. Luděk Žalud, Ph.D.

BRNO 2022

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study program **Cybernetics, Control and Measurements**

Department of Control and Instrumentation

Student: Bc. Miloš Cihlář ***ID:*** 201303

Year of study: 2 ***Academic year:*** 2021/22

TITLE OF THESIS:

Simulation of Robotic Search of Lost Radiation Sources

INSTRUCTION:

1. Make search for mobile robotics simulation tools with a focus on Gazebo and Ignition.
2. Familiarize yourself with Robot Operating System 2.0 and summarize its advantages and possible disadvantages.
3. Based on the results of the semester work, select and implement in the chosen simulation tool two suitable algorithms for searching for so-called lost radiation sources in the external environment together with the supervisor.
4. Compare the implemented algorithms in the supplied map and also compare their advantages and disadvantages.

RECOMMENDED LITERATURE:

Mahtani Anil, Effective Robotics Programming with ROS, Packt Publishing Limited, 2016, ISBN 9781786463654

Date of project specification: 7.2.2022 ***Deadline for submission:*** 18.5.2022

Supervisor: prof. Ing. Luděk Žalud, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
Chair of study program board

ABSTRACT

The master thesis considers simulating the outdoor environment and lost radiation sources in the Gazebo simulator and then deals with radiation search algorithms. Simulators play an essential role in research, and robotics companies widely use them. Simulation accelerates and facilitates testing, verification, and evaluation of developed algorithms. Therefore the thesis establishes a system based on ROS2 and Gazebo that simulates robot models. The performance of the simulation is evaluated. The work presents and compares a model of a four-wheel skid steering mobile robot with the model created in the simulator. In future work, this model comparison allows the improvement of the robot model simulation's parameters based on the real model tested. A solution is proposed for the trajectory tracking problem based on linear proportional-integral (PI) or optimal linear quadratic regulator (LQR). One method of modeling radiation sources and sensors to simulate radiation was suggested. The simulated radiation is compared with the measurement gained from a real (and simulated) outdoor experiment, in which the robot searches the radiation source. The thesis describes the particle filter designed for searching a lost radiation source.

KEYWORDS

ROS2, Gazebo, Simulator, Radioactivity, Radiation sensor, Lost radiation source, Gazebo environment, Skid steering mobile robot, Particle filter, Motion analysis, Dynamic analysis, Dissipative forces, Radioactivity simulation

ABSTRAKT

Simulátory, společnostmi zabývající se robotikou hodně využívané, hrají důležitou roli při výzkumu robotů. Zrychlují, zjednodušují, zlevňují a usnadňují vývoj softwaru a algoritmů. Magisterská práce se proto zabývá návrhem systému, založeného na ROS2 a Gazebo simulátoru, umožňující simulaci pozemních robotů ve vnějším prostředí s možností hledat ztracené radiační zdroje. Práce navrhuje několik metod vytváření prostředí v Gazebo simulátoru včetně návrhu prostředí z mračna bodů a je vytvořen model čtyřkolového, smykově řízeného mobilního pozemního robota. Chování robota v simulátoru bylo ověřeno a upraveno pomocí teoretického dynamického popisu robota. Před simulací algoritmů pro hledání ztracených radiačních zdrojů je navržena metoda sledování referenční trajektorie pomocí proporcionálně integračního (PI) a lineárně kvadratického (LQ) regulátoru a navrhuje metodu k simulaci zdroje radiace a jeho měření. Hledání radiačního zdroje jsou použity dvě typově odlišné metody, kdy jedna je založena na prozkoumání celé oblasti a vytváří mapu radiace, a druhá metoda založená na částicovém filtru aktivně hledá ztracený zdroj záření.

KLÍČOVÁ SLOVA

ROS2, Gazebo, Simulátor, Radioaktivita, Senzor radiace, Ztracený radiační zdroj, Prostředí v Gazebu, Smykově řízený mobilní robot, Částicový filtr, Analýza pohybu, Analýza dynamiky, Odporové síly, Simulace radioaktivity

ROZŠÍŘENÝ ABSTRAKT

ÚVOD

Robotika je multidisciplinární obor kombinující znalosti z elektrotechniky, informatiky, matematiky, fyziky, řízení, měření, umělé inteligence, filosofie, psychologie a dalších vědních oborů, kterých se robotika, jako obor, dotýká. V posledních dvaceti letech je robotika, zejména mobilní robotika, na velkém vzestupu. Moderní robotické systémy vybaveny množstvím různých senzorů, aktuátorů, počítačů a dalším vybavením mají za cíl splnit požadovanou misi, úkol, kterou může být libovolná činnost vykonávána člověkem nebo činnost, jež člověk vykonávat nechce či nemůže. Robot, způsobilý vykonávat tyto úkoly, musí být schopen základního pohybu, orientace, vnímání a rozhodování. Všechny zmíněné obory, jimiž se robotika zabývá, pomáhají k rozvojí schopností robota.

Senzory jsou drahé, práce s hardwarem náročná, riziko nehody velké, proto vzniká tlak a tendence přesouvat vývoj, testování a lazení algoritmů do simulátorů, které značně urychlují, zjednodušují a zlevňují vývoj a výzkum robotů.

Tato diplomová práce se zabývá simulací kolových mobilních robotů a různými metodami vytváření a simulování vnějšího (outdoor) prostředí, včetně prostředí vytvořeného na základě změřených dat. Práce dále porovnává model čtyřkolového mobilního robota se simulací. Na základě modelu jsou navrženy různé metody řízení a sledování trajektorie. Závěrečným cílem celé práce je otestovat algoritmy hledající ztracené radiační zdroje, včetně simulace radioaktivity simulátoru Gazebo.

SIMULÁTOR

Simulace poskytuje vývojářům a vědcům řadu výhod, především zrychlení, zefektivnění, zkvalitnění a usnadnění práce. Proto taky přibližně 70% organizací zabývající se robotikou používají simulátory. Simulátory mají taky určitá omezení, která se týkají především výpočetní náročnosti. Byl zkoumán vliv na výpočetní náročnost vzhledem k počtu robotů v simulaci. Bylo zjištěno, že využití paměti roste lineárně s množstvím simulovaných robotů, ale využití procesoru při rtf = 1 (real time factor) kvadraticky roste. Práce využívá robotický simulátor Gazebo, ve kterém lze simulovat vnitřní a vnější prostředí. K vytvoření vnějšího prostředí ze změřených dat byl použit point cloud, získaný z experimentů popsaných ve ve článku An automated heterogeneous robotic system for radiation surveys: Design and field testing [44] z roku 2020. Původní point cloud obsahuje přes 29000000 bodů rozmístěných na oblasti 200m x 170m. Před vytvářením 3D modelu musel být původní point cloud redukován na 2000000 bodů a došlo k vyfiltrování neúplných objektů, převážně stromů. Z takto připraveného point cloudu pomocí nástrojů Meshlab and Cloudcompare byl vytvořen 3D model a extrahována textura.

MODEL ROBOTA

K hledání ztracených radiačních zdrojů byl vybrán a simulován robot Orpheus X4. Robot se v Gazebo simulátoru skládá ze čtyř kol a těla. Fyzikální vlastnosti robota ovlivňují hlavně základní parametry, jako je hmotnost, rozložení hmotnosti, velikost robota. Tyto parametry jsou definovány při vytváření objektů (tvar kol a těla) modelu. Tření kol a kloubů ovlivňuje pohyb robota. Pro klouby (spojení mezi tělem a koly, pohon) je nastaveno statické a dynamické tření. Pro kola je nejdůležitější tření a maximální síla, kterou je robot odpuzován od styku s povrchem, ovlivňuje skákavost robota, a maximální množství kontaktů.

Po otestování chování robota v simulaci je popsán čtyřkolový mobilní robot z pohledu kinematického a dynamického. Na základě těchto modelů lze lépe popsat chování robota v simulátoru vzhledem k nastavovaných parametrům, což je možné využít při porovnávání pohybu skutečného a simulovaného robota.

Pro splnění cíle práce, tedy testování algoritmů na hledání ztracených radiačních zdrojů, je zapotřebí vyřešit problém sledování trajektorie. Pro navigaci přes množinu waypointů je nejprve vygenerována trajektorie pomocí kubického spinu (Cubic spline), zajišťující spojitost prvních dvou derivací v průjezdních bodech. Regulační problém je rozdělen na dvě části, první, vnitřní zpětnovazební smyčka, řídí lineární a úhlovou rychlost robota pomocí PI nebo optimálního LQ regulátoru. Druhá generuje referenční rychlosti na základě současné pozice robota vygenerované trajektorie.

SIMULACE RADIOAKTIVITY

Práce a testování algoritmů na reálných robotech je drahé, pomalé a složité, o to více to platí pro zdroj radiace. Experimenty se zdroji radiace musí splnit všechny legislativní podmínky, což prodlužuje a komplikuje jejich přípravu. Jedním z cílů práce je umožnit simulovat radiační zdroje a senzory. Práce využívá existující řešení popsané v článku *Simulating Ionising Radiation in Gazebo for Robotic Nuclear Inspection Challenges*, které je modifikováno pro použití s ROS2 a více scintilačními senzory s různými útlumy v různých překážkách pro každý zdroj.

HLEDÁNÍ ZTRACENÝCH RADIAČNÍCH ZDROJŮ

Již více než před 120 lety Henri Becquerel, Pierre Curie a Marie Curie-Skłodowska objevili a popsali radioaktivitu. Později v roce 1942 byl spuštěn první jaderný reaktor v Chicagu v USA. O tři roky později byl proveden první test jaderné zbraně. V dnešních dnech se radioaktivita používá v mnoha různých oborech, od průmyslu, armády, lékařství až po zemědělství. S rostoucím použitím radioaktivních látek, zároveň roste potřeba bezpečně hledat radioaktivní materiál.

Poslední kapitola se zabývá simulací algoritmů pro lokalizaci radiačního zdroje.

První metoda je založena na průzkumu terénu pomocí předem navržených trajektorií a současným snímáním radiace. Výstupem je tedy mapa obsahující intenzity záření. Z mapy je později určena pozice, případně další parametry zdroje. Nevýhodou tohoto přístupu je dlouhá doba mapování pro větší oblasti, případně úplná nemožnost prozkoumat celé území. Proto dalším simulovaným algoritmem je částicový filtr (Particle filter), který aktivně vyhledává pozici zdroje. Jeho výhodou je rychlost, ovšem není tak přesný jako přístup založený na tvorbě mapy, a nemusí konvergovat.

Author's Declaration

Author:	Bc. Miloš Cihlář
Author's ID:	201303
Paper type:	Master's Thesis
Academic year:	2021/22
Topic:	Simulation of Robotic Search of Lost Radiation Sources

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno 17. 05. 2022 .....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to express my thanks to the advisor of my thesis, prof. Ing. Luděk Žalud Ph.D. for his valuable comments and excellent leadership. I would like to also thank Zuzana Urbancová for her never-ending support. And last but not least, I would like to express my gratitude to my family, who helped me and supported me all the time in my studies.

Brno 17. 05. 2022 .....

author's signature

Contents

Introduction **14**

1 Preliminaries **16**
1.1 Source of Radiation . . . 16
1.1.1 Isotopes . . . 18
1.1.2 Radioactivity . . . 19
1.1.3 Units of Radioactivity . . . 23
1.1.4 Dosimetry . . . 24
1.1.5 Health Risks . . . 26
1.1.6 Radiation Protection . . . 27
1.2 ROS and ROS2 . . . 28
1.3 Robotic Simulators . . . 29
1.3.1 Robotic Simulators Overview . . . 31
1.3.2 Gazebo . . . 32
1.3.3 Ignition Gazebo . . . 35

2 Environment **37**
2.1 Environment from Model Database . . . 39
2.2 Generation of Random Environment . . . 40
2.3 Building Editor . . . 41
2.4 Environment from Real Data . . . 41
2.4.1 Digital Elevation Model . . . 42
2.4.2 Point Cloud . . . 43

3 Robot Models **46**
3.1 Orpheus X4 . . . 46
3.2 Robot Model in Gazebo . . . 47
3.2.1 Methods for Defining Models . . . 47
3.2.2 Physical Properties . . . 48
3.3 Motion Analysis . . . 51
3.4 Dynamic Model . . . 53
3.4.1 Dissipative Forces . . . 56
3.4.2 Comparing with Gazebo . . . 61
3.5 Controllers . . . 66
3.5.1 Trajectory Generator . . . 67
3.5.2 Trajectory Controller . . . 69
3.5.3 Velocity Controller . . . 70

3.5.4 Trajectory Tracking 74
3.6 Coordinate System 78
3.6.1 REP 105 78
3.6.2 Conversion of Geographic Coordinate System 79

4 Radioactivity Simulation **81**
4.1 Radiation Gazebo Plugin 81
4.1.1 Gazebo Experiment 82

5 Searching Lost Radioactive Source **86**
5.1 Mapping Based Approach 86
5.2 Particle Filter 89

Conclusion **95**

Bibliography **96**

List of Figures

1.1 The division of EMR . . . 16
1.2 An overview of ionizing radiation . . . 17
1.3 Table of nuclides and their half-life . . . 19
1.4 The energy spectrum of emitted electrons and positrons during β decay 22
1.5 Architecture of ROS1 and ROS2 in terms of their layers . . . 29
1.6 A reason for using robotic simulations . . . 30
1.7 Types of commonly used robotic simulators . . . 31
2.1 The Gazebo simulator's Real-Time Factor . . . 37
2.2 CPU and memory usage of simulator . . . 38
2.3 The evaluation of the Gazebo simulator . . . 38
2.4 Environment from model database . . . 39
2.5 Overview of distribution option in the population of models . . . 40
2.6 The difference between DTM and DSM. . . . 41
2.7 Create an environment with LIRS-WCT . . . 42
2.8 Two types of real data environment. . . . 43
2.9 An example of point cloud reduction . . . 44
2.10 Using the output density . . . 45
3.1 Orpheus X4 . . . 46
3.2 The robot's center of mass . . . 48
3.3 Parameter minDepth and its visualization in Gazebo . . . 49
3.4 Parameter maxContacts and its visualization in Gazebo . . . 49
3.5 Robot's bounciness . . . 50
3.6 The robot trajectory depending on maxContacts parameter . . . 50
3.7 Robot coordinates system. . . . 52
3.8 Robot wheels' velocities . . . 53
3.9 Forces performing the wheel motion . . . 54
3.10 Simulink scheme of longitudinal velocity . . . 55
3.11 Simulink scheme of angular velocity . . . 56
3.12 Approximation of Signum Function by Arctangent . . . 57
3.13 Approximation of Joint Static Resistive force with Hyperbolic Tangent 58
3.14 Comparison of linear velocities model 3.11 and robot in the Gazebo simulator . . . 63
3.15 $R2$ and $RMSE$ values for linear velocity . . . 63
3.16 Comparison of linear velocities model 3.12 3.31 and robot in the Gazebo simulator . . . 64
3.17 $R2$ and $RMSE$ values for angular velocity with Coulomb friction model 64

3.18 Comparison of linear velocities model 3.12 3.32 and robot in the Gazebo simulator . . . 65
3.19 $R2$ and $RMSE$ values for angular velocity with a combination of Coulomb and viscous friction model . . . 65
3.20 Trajectory tracking diagram . . . 66
3.21 Diagram of the trajectory generator . . . 67
3.22 Example of cubic spline trajectory . . . 69
3.23 Feedback loop with PI controller . . . 71
3.24 Linear and angular velocity response - PI . . . 72
3.25 Feedback loop with LQR . . . 73
3.26 Linear and angular velocity response - LQR . . . 74
3.27 Simulink diagram of T-PI trajectory tracking controller . . . 75
3.28 T-PI: Desired and actual (true) trajectory and linear and angular velocity . . . 75
3.29 Simulink diagram of LQ-PI trajectory tracking controller . . . 76
3.30 LQ-PI: Desired and actual (true) trajectory and linear and angular velocity . . . 76
3.31 Simulink diagram of LQ-LQ trajectory tracking controller . . . 77
3.32 LQ-LQ: Desired and actual (true) trajectory and linear and angular velocity . . . 77
3.33 Multi-robot relationship between frames in ECEF . . . 78
3.34 Earth-centered, Earth-fixed coordinate system . . . 79
4.1 The decay scheme of Cobalt-60 . . . 82
4.2 Using the output density . . . 83
4.3 The heat map of radiation (ideal sensor) . . . 84
4.4 The heat map of radiation (real sensor) . . . 85
5.1 Mapping-based algorithm example . . . 86
5.2 Mapping-based radiation search real-world experiment . . . 87
5.3 Mapping-based radiation search experiment in simulation . . . 87
5.4 Particle filter - first experiment . . . 91
5.5 Particle filter - second experiment . . . 92
5.6 The example of non-convergence of particle filter . . . 93
5.7 Particle filter - third experiment . . . 93

List of Tables

1.1 Examples of quality factor Q_R and tissue weighted factor w_T 26
1.2 Quality factor Q_R for 26
1.3 Tissue weighted factor w_T 26
3.1 Experiments overview 62
5.1 Particle filter experiment overview 91

Introduction

Robotics is an interdisciplinary field that involves electrical engineering, computer engineering, mathematics, information technology, control engineering, physics, etc. Mobile robotics has been in rapid development in recent years. Robotic systems are equipped with numerous sensors and perform a variety of tasks. The main functions in robotics are localization and mapping to determine the exact robot's position. Path planning searches trajectory from start to final position. Motion control, computer vision, perception, artificial intelligence, etc., are challenges that robots must complete. Therefore robots are complex hardware devices, and their development is expensive. Simulation is the way how to accelerate development, especially algorithm testing. There is no risk of breaking robots or very expensive sensors and equipment with the simulator. With correctly modeled robots and simulation environments, the software used in the simulation can be directly used in the real world [1]. One of the ways how to design a robot's environment in a simulator is to use real sensors data. The article *Tool for 3D Gazebo Map Construction from Arbitrary Images and Laser Scans* [2] deals with constructing a map from the laser scans. Improvement previous work *Automatic tool for Gazebo world construction: from a grayscale image to a 3D solid model* [3] is construction 3D Gazebo world using a grayscale image.

Using robots in hazardous environments can prevent and reduce the human health risk. One of the hazardous is nuclear environments. Most of the organizations dealing with robot development use a simulators [32]. It is advantageous to deal with implementation and modeling radiation sources and sensors. The work *Localization of Ionizing Radiation Sources by Cooperating Micro Aerial Vehicles With Pixel Detectors in Real-Time* [4] deals with the Timepix semiconductors detector and its simulation in the Gazebo simulator. And the next work *Simulating Ionising Radiation in Gazebo for Robotic Nuclear Inspection Challenges* [5] implements radiation sensors more generally. The simulated environment has a number of benefits.

The article *An automated heterogeneous robotic system for radiation surveys: Design and field testing* [44] presents a multi-robotic system designed for radiation mapping and source localization. With properly designed simulation and models, the whole experiment can be realized easier multiple times.

This master thesis deals with creating a system based on ROS2 and the Gazebo simulator that allows to simulate, test, and evaluate robotics software. The work offers several ways to model the environment, including the world construction from the real-based world. The robot model is presented the same as the theoretical robot model compared with the simulation. Movement robot into trajectory is necessary for proper simulation of searching radiation sources. Therefore, the method

is proposed to solve the trajectory tracking problem based on PI and LQ controllers.

The thesis is organized as follows. In Preliminaries section are mentioned the theoretical information about the source of radiation, the ROS and ROS2 are compared, and the possibilities of different simulators are mentioned. The section Environment discusses the influence number of robots on Gazebo performance, and several options are described to create simulated worlds. In the third chapter Robot Models, the robot model is designed, and theoretical model descriptions are derived. In the same chapter the trajectory tracking problem is also solved. The Radioactivity Simulation chapter is described and tests radiation sources and sensors. Furthermore, the two last chapters Searching Lost Radioactive Source, Conclusion contain the radiation search algorithm and work conclusion.

1 Preliminaries

1.1 Source of Radiation

A *radiation source* is an object, which can produce energy embodied by waves or particles, and these waves or particles can transmit through space or materials. The radiation can be classified into two following sections, as the picture below refers 1.1, according to the radiation energy. The first is that *non-ionizing*[1] radiation includes *electromagnetic radiation* (EMR), such as *radio waves*, *microwaves*, *infrared* and *visible light*, and these waves do not have sufficient energy to ionize atoms or molecules.

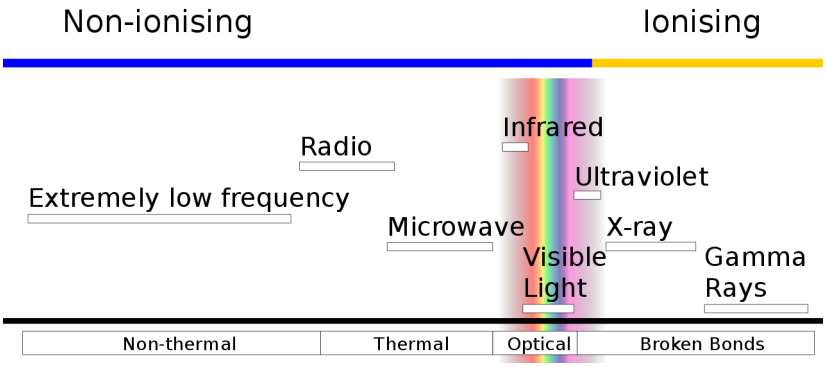


Fig. 1.1: The picture shows the division of EMR into non-ionizing and ionizing radiation. For each group is shown the type of waves with their immediate physical effects. [7]

The second is *ionizing* radiation, and this thesis deals with only ionizing radiation that has multiple health risks than non-ionizing radiation. Moreover, therefore the simulation of radiation sources and sensors is one of the aims of this thesis. Better division for ionizing radiation is according to Radiation Detection and Measurement [8]. All of the following sections in Source of Radiation section are taken over from the book [8] unless otherwise stated.

Radiation is a process when occurs the emission of waves or particles mainly originates in the atomic or nuclear process. The radiation can be categorized into four following general parts according to what is emitted.

- **Charged particulate radiation**
  - Fast electrons

[1] Ionization is a process when the atom or molecule gains an electrical charge by losing or gaining an electron.

  - Heavy charged particles
- **Uncharged particulate radiation**
  - Electromagnetic radiation
  - Neutrons

Fast electrons, in other words, *beta radiation* or *beta ray* is a beam of beta particles that includes electrons or positrons, which is produced in nuclear decay or any other atomic process. This radiation is described in more detail in section Beta Decay.

The next type of charged radiation is heavy charged particles. The aforementioned radiation is a group of radiation that encompasses all energetic ions with a mass of one atomic unit or grater. Namely, it is *alpha particles* [2], which is explained more deeply in section Alpha Decay, protons or fission products.

The electromagnetic radiation mentioned above is the first uncharged radiation include *X-rays* and *gamma rays*. The difference between *X-rays* and *gamma rays* is in their source. *X-rays* are emitted by rearranging electron shells of atoms, whereas gamma rays are produced from transition within the nucleus itself.

Neutron radiation is a beam of free neutrons that can react with other nucleus and cause new isotopes. Neutrons are obviously created when an atom is split in nuclear fission.

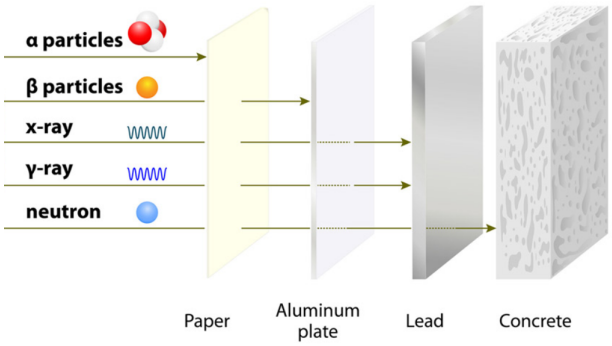


Fig. 1.2: An overview of ionizing radiation and its ability to penetrate different types of obstacles. [9]

The range of energy, which is carried by ionizing radiation, is over six decades. The minimum energy is set to bound when radiation can produce ionization in typical material. The radiation and their energy differ in the ability to penetrate thickness of material. In other words, the radiation can be split according to its "hardness".

[2] Alpha particles ($^4_2He^{2+}$) consist of two protons and two protons joined together (without two electrons).

Soft radiation can penetrate only a small thickness of a material such as a sheet of paper or human skin. It encompasses alpha particles (1.2) or low-energy X-rays. The previously mentioned radiation sources must be deposited in thin layers. Thicker sources can be subject to *self-absorption*, which affect energy released from the surface.

Beta particles are more penetrative than alpha particles, and it can pass through units centimeters of material depending on energy. Harder radiation, gamma, high-energy, X-rays, or neutron can go through millimeters or centimeters materials without changing physical properties.

Before radioactivity is specified, it is necessary to mention information about what *isotopes* are.

1.1.1 Isotopes

If *radioactivity* is mentioned, the following terms are often used. When talking about atomic nuclei and their properties, the nuclei are often called *nuclide*. Atomic nuclei consist of protons and neutrons. *Atomic number* or sometimes called *proton number* indicates the number of protons in the nucleus and is marked as Z. The same way is denoted *neutron number* N, which holds the number of neutrons. The sum of *atomic number* and *neutron number* is the *mass number*.

$$A = Z + N \tag{1.1}$$

Therefore, the mass number includes the number of *nucleons*, which is a general designation of protons and neutrons. The better way how to note chemical elements with respect to the number of protons and neutrons is to write ${}^{A}_{Z}X$, where X is chemical element.

Isotope is a modification of chemical elements that differ in the number of neutrons in a nucleus. So, isotopes are nuclide with the same atomic number Z and with the different neutron number N.

All isotopes with the same number of protons (Z) and electrons are *neutral atoms* and are placed in the periodic table. *Neutral atoms* have identical chemical properties, but the nuclear properties of various isotopes are much different. All isotopes are organized in the *table of nuclides* which is shown in figure 1.3.

The *table of nuclides* shows a layout of isotopes of all chemical elements. On the horizontal axis is proton number Z, which defines chemical elements, and on the vertical axis is shown neutron number. A color of nuclides on the chart 1.3 assigns their half-life. Half-life is a time that is required the disintegration of one-half of atomic nuclei [12]. As it graph follows, most of a nuclides are unstable and are placed around stable elements.

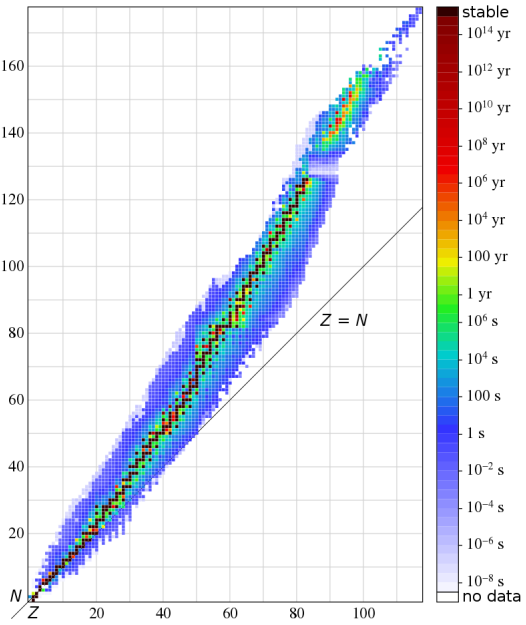


Fig. 1.3: Table of nuclides and their half-life [11]

The light stable nuclides lie near the line $Z = N$ and have a same number of protons and neutrons. On the contrary heavy nuclides have more neutrons than protons and lie over the line $Z = N$. The last and heaviest stable isotope is a Bismuth-209[3] ${}^{209}_{83}Bi$.

This entire chapter is taken from [10].

1.1.2 Radioactivity

The physical property of unstable isotopes is *radioactivity*. *Radioactivity*, nuclear decay, or radioactive decay is a process when isotopes lose energy and tend to a more stable configuration (unstable nuclide is converted to another nuclide). As stated above, this process emits radiation appearing as charged particles, electromagnetic waves, or neutrons to space. Radioactive decay is a stochastic process that is described with two significant terms, *activity* and *half-life*.

The *activity* $A_c$of an unstable isotope (radioisotope) is defined as

$$A_c = \frac{dN}{dt} = -\lambda N \tag{1.2}$$

where N is a number of radioactive nuclei. Disintegration speed dN/dt is directly proportional to N. Lambda, defined as *decay constant* with unit s^{-1}. Activity

[3]Bismuth-209, according to [13], undergoes alpha decay and has a half-life of $(1.9 \pm 0.2) \times 10^{19}$ yr.

measures disintegration rate, but it is not equal to the emission rate of radiation sources. In many cases, radiation is made only a fraction of all decay. To determine the emission rate from an *activity* is necessary *decay scheme* of a particular isotope.

When *activity* is used in a specific sample, the *specific activity* can be defined as *activity* per unit of mass of the radioisotope.

$$A_s = \frac{activity}{mass} = \frac{\lambda N}{N\frac{M}{A_v}} = \frac{\lambda A_v}{M} \tag{1.3}$$

where M is *molecular weight* of the radioisotope, A_v is Avogadro's number ($A_v = 6.02 \times 10^{23}$). *Decay constant* lambda is evaluated

$$\tau = \frac{ln(2)}{\lambda} \tag{1.4}$$

where τ is half-life in seconds s.

The unit used for the measurement of radiation energy is *electron volt* denoted as eV. *Electron volt* is defined as kinetic energy of an electron which is accelerated by a potential difference of 1 volt. For measurement energies for ionizing radiation, *kiloelectron volt* (keV) and *megaelectron volt* (MeV) are commonly used. *Electron volt* is not the SI unit. For conversion from *electron volt* to *joule* (J) is used following equality.

$$1eV = 1.602 \times 10^{-19} J \tag{1.5}$$

Photon energy of gamma or X- rays is directly proportional to the frequency as follows

$$E = hv \tag{1.6}$$

where h is Planck's constant ($h = 6.626 \times 10^{-34} Js$ or $4.135 \times 10^{-15} eVs$) and v is frequency in Hz.

Alpha Decay

From isotopes are released *alpha particles*, consisting of two protons and two neutrons when *alpha decay* occurs. And without a product of rare *spontaneous fission*, alpha particles are the heaviest product of nuclear decay. *Alpha particles* can be denoted as the nucleus of ${}^4He$ or ${}^4_2\alpha$. Because of the lack of electrons, alpha particles have a strong positive charge and are highly reactive with surrounding matter. According to the following equation [14],

$${}^A_N X_N \rightarrow {}^{A-4}_{N-2} X'_{N-2} + {}^4_2 He_2 \tag{1.7}$$

an unstable nucleus X disintegrates into lighter nucleus X' and the aforementioned *alpha particles*. For example [10],

$$^{238}_{92}U_{146} \rightarrow ^{234}_{90} Th_{144} + ^{4}_{2} He_2 \qquad Q = 4.25MeV \tag{1.8}$$

The probability of released alpha particles is governed by *quantum-mechanical barrier penetration*, described in Modern Physics [14].

During the nuclear decay process, energy is released, distributed as kinetic energy between *alpha particles* ^4He and the nucleus X'. Energy is obtained from atomic mass differences of products of nuclear decay according to [14]

$$Q = [m(X) - m(X') - m(^4He)]c^2 \tag{1.9}$$

And the energy of alpha particles is determined [14] as

$$K_\alpha \cong \frac{A-4}{A} Q \tag{1.10}$$

Beta Decay

Beta decay is another nuclear disintegration process. Neutron in the nucleus is changed into a proton[4] and proton into a neutron[5]. Therefore atomic and neutron number (Z and N) change by one unit, but mass number A is unaffected. There are two variants of *beta decay.*[14] As was mentioned before, beta radiation is a beam of electrons and positrons.

Before the *beta decay* is specified, it is necessary to describe what *neutrino* is. *Neutrino* ν and its antiparticle *antineutrino* $\bar{\nu}$ is elementary particle. Its mass is low compared to other elementary particles. The electric charge of neutrino is zero and is nonreactive. First of all, he assumed the existence of a neutrino Wolfgang Pauli in 1930. [10]

The first type of decay process is when a neutron is changed according to the following equation. All followings information in Beta decay section are derived from [14].

$$n \rightarrow p + e^- + \bar{\nu} \tag{1.11}$$

Neutron n is changed to protons p, electrons e^-, and antineutrino when beta decay occurs. Next equation of nuclear decay if for $\beta-$ radiation.

$$^{A}_{Z}X_N \rightarrow ^{A}_{Z+1} X'_{N-1} + e^- + \bar{\nu} \tag{1.12}$$

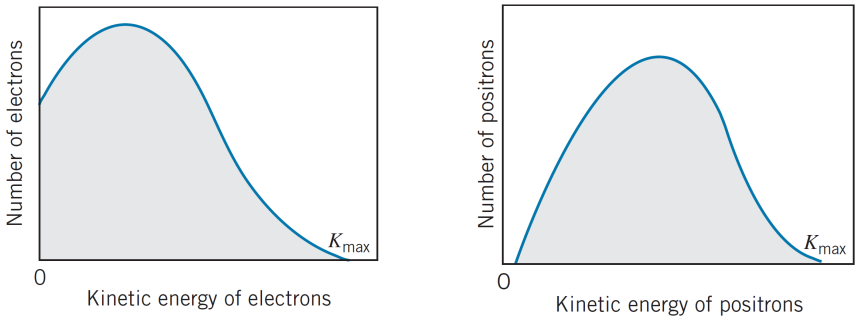
[4] And electron with antineutrino

[5] And positron with neutrino

Energy released during beta decay is distributed between energy of antineutrino E_v, the kinetic energy of electron K_e, and negligible kinetic energy of $K_{x'}$.

$$Q = E_\nu + K_e + K_{X'} \cong E_\nu + K_e \tag{1.13}$$

Unlike beta disintegration, the energy of an emitted particle has a continuous energy spectrum shown in figure 1.4a. Energy of electron is from zero up to K_{max}.



(a) The energy spectrum of electrons emitted in beta decay.

(b) The energy spectrum of positrons emitted in beta decay.

Fig. 1.4: The energy spectrum of emitted electrons and positrons during β decay

Another *beta decay* process is when a proton is changed to neutron, *positive electron* (*positron*), and a *neutrino*, as shown by the following equation.

$$p \rightarrow n + e^+ + \nu \tag{1.14}$$

Nuclear decay equation for $\beta+$ radiation is shown bellow.

$${}^{A}_{Z}X_N \rightarrow {}^{A}_{Z-1}X'_{N+1} + e^+ + \bar{\nu} \tag{1.15}$$

And its energy spectrum is shown in figure 1.4b.

Electromagnetic Radiation

After Alpha Decay and Beta Decay, gamma radiation is the next type of radiation. Gamma radiation is high excited photons without any charge; therefore is more dangerous than the aforementioned radiation. The absence of charge causes it's dangerous because it goes through air, skin, and wood. Electromagnetic radiation can originate in many ways. Gamma radiation is emitted from atomic nuclei. The photons are released when the nuclei transit from higher to lower-laying nuclear levels. One of the first causes is connected to beta decay.

The beta decay with a long half-life time causes an excited state of the daughter nucleus. With a much shorter half-life time, the subsequent transitions release

gamma radiation whose energy is equal to the difference in energy between nuclear states. Annihilation radiation is another origin of gamma radiation. When the nucleus undergoes $\beta+$ decay, the positrons are emitted in the primary decay process. The energy of released positrons is very low. They travel a few millimeters before recombining with normal negative electrons. The collision with an antiparticle is called annihilation. The annihilation results are the disappearance of both the original particle and antiparticle and are replaced with two photons. *Bremsstrahlung* is the process causing releasing photons from the kinetic energy of fast electrons, which are decelerated by another particle, such as atom nuclei.

Neutron Radiation

Neutron radiation is a beam of free neutrons that are released by spontaneous or nuclear fission. The neutron does not have a charge, and it can interact with nuclei of other atoms and turns it into new isotopes, which can turn into a new radiation source. This is one of the reasons for high radiation hazards for tissue.

1.1.3 Units of Radioactivity

The activity of a radioisotope is a rate of decay described in the section Radioactivity. The historical unit of activity is *curie* (Ci). *Curie* is defined as the activity (disintegration per second) of 1 gram of pure ^{226}Ra. Its value is evaluated as

$$1Ci = 3.7 \times 10^{10} s^{-1} \tag{1.16}$$

For laboratory use, the units *millicurie* (mCi) and *microcurie* (μCi) are more suitable.

At the General Conference of Weights and Measures (GCPM) in 1975, *curie* the standard unit for activity was replaced by *becquerel* (Bq). *Becquerel* is defined as one disintegration per second.

$$1Bq = 2.703 \times 10^{-11} Ci \tag{1.17}$$

The radiation source can also be expressed by *luminosity* L, which describes the power of emitted radiation. *Luminosity* has units the watt (W). The watt is defined as the joule per second

$$W = J \cdot s^{-1} = kg \cdot m^2 \cdot s^{-3} \tag{1.18}$$

and therefore, finding the power of the radiation source is necessary to know the energy of the radiation carrier.

The power of the radiation source captured by the unit area is the intensity I or radiation flux [15].

$$I = \frac{L}{4\pi r^2} \tag{1.19}$$

where r is the distance from the radiation source. Unit of intensity I is $W \cdot m^{-2}$.

Similarly, as *luminosity* per area unit, the activity per area unit can be defined as

$$I_a = G\frac{A_c}{4\pi r^2} \tag{1.20}$$

where G (dimensionless) is a number of particles released from radionuclide per one nuclear decay. Unit of I_a is $s^{-1} \cdot m^{-2}$.

1.1.4 Dosimetry

Various kinds of radiation affect tissue (especially human tissue). See chapter Health Risks. Sources of radiation are also naturally located in our environment. First, Cosmic rays continuously impact the Earth and interact with Earth's atmosphere and create radioactive isotopes ^{14}C, Tritium ^{3}H or ^{7}Be. Next, radiation sources are placed in the Earth's crust. [17] Besides natural sources, there are extensive usages of radiation in many fields.

Therefore was a need to create units to describe the properties and effects of radiation on the tissue because the damage caused by radiation is not easily quantified. The dosimetry was originally used to determine the integrated energy deposited to a person working in a radiation environment. In general, dosimetry deals with a dose and dose rate to refer to integrated energy and the energy deposited per unit time.

In the damage detection of radiation plays a role in two essential aspects. First, the damage depends on the amount of radiation absorbed by a material, and the amount of radiation has a crucial effect when acute tissue damage appears. Second, the stochastic effects do not depend on radiation intensity. The mutation of cells, one of the stochastic effects, can cause damage when tissue absorbs even low radiation levels.

Roentgen

Roentgen R measures only radiation based on photons only (X-rays and gamma radiation). Value of $1R$ is defined as 2.58×10^{-4} coulombs of a positive and negative charge, which is caused by radiation in one kilogram of air. *Roentgen* is not used at dosimetry because it is not applicable to all tissue.

Absorbed Dose

Absorbed dose is described as the amount of energy deposited to medium with unit mass by ionizing radiation. The unit of the absorbed dose is *Gray* with the abbreviation Gy.

$$1Gy = 1J \cdot Kg^{-1} \tag{1.21}$$

The older and still used unit of the absorbed dose is *rad* which is equivalent to $0.01Gy$.

Equivalent Dose

The *absorbed dose* described above 1.1.4 cannot determine the biological effect of radiation, and it only brings information about the quantity of energy absorbed by a material. The absorbed dose does not differentiate types of radiation and deals with all kinds of radiation equally.

The dosimetry focuses on safety, and therefore, equivalent doses characterize radiation's effect on tissue.

$$H_{T,R} = w_R \cdot D_{T,R} \tag{1.22}$$

The equivalent dose $H_{T,R}$ caused by radiation type R is gained as absorbed dose $D_{T,R}$ multiplied by a radiation weighting factor w_R. The weighted factor is

$$w_R = Q_R \cdot N_R, \tag{1.23}$$

where Q_R is *quality* factor and N_R is *modified* factor for the radiation R. N_R is one for external[6] sources. The formula above 1.22 is only for one type of radiation. For mixed radiation

$$H_T = \sum_R w_R \cdot D_{T,R} \tag{1.24}$$

Unit of the equivalent dose is the same as the absorbed dose $J \cdot Kg^{-1}$, but for distinguishing its specific name is *sievert* with symbol Sv.

Effective Dose

The relationship between the absorbed dose and the equivalent dose is similar to the relationship between the equivalent dose and a *effective dose*. Equivalent dose distinguish different radiation types for one tissue type, but the sensitivity of various tissue types to the same kind of radiation is different. The effect of a different type of radiation on various tissue is described by effective dose E with an equation.

$$E = w_T \cdot H_T, \tag{1.25}$$

[6]Radiation comes from around the environment [18]

where w_T is the tissue weighting factor, and H_T is the equivalent dose in the tissue T.

In the following table 1.1 are shown quality factor Q_R and tissue weighted factor w_T.

Tab. 1.1: Examples of quality factor Q_R and tissue weighted factor w_T

Tab. 1.2: Quality factor Q_R for

Type of radiation	$\boldsymbol{Q_R}$ [-]
α particles	20
β particles	1
X and gamma rays	1
Neutrons (various energy)	5-20

Tab. 1.3: Tissue weighted factor w_T

Tissue or organ	$\boldsymbol{Q_R}$ [-]
Gonads	0.2
Colon, Lung, Stomach	0.12
Bladder, Breast, Liver	0.05
Skin, Bone surface	0.01

More information about dosimetry can be found in Physics & Engineering of Radiation Detection [16]. From this book is inspired section Dosimetry unless otherwise stated.

1.1.5 Health Risks

The radiation can interact with atoms (more information in [16]) as same as with atoms witch are in all organism's cells. The cell consists of various organelles, and not of them has the same critical role in cell functioning. The most important are the chromosomes which include DNA, and when the chromosomes are damaged, it has the potential to cause wrong cell behavior. No matter how intense ionizing radiation is (every intense ionizing radiation can cause damage) but damage occurrence probability decreases with decreasing radiation. Acute exposure occurs when tissue receives a high instantaneous dose in a short time. The amount of dose is generally intended to $0.1Gy$ with time hours or a few days. The cells start to die when acute exposure occurs. Final consequences for organisms depend on dose size.

- Dose $> 2Gy$: hair loss or reddening skinn
- Dose $> 5Gy$: visual impairment and cataract
- Dose $> 25Gy$: deep skin necrosis
- Dose $> 50Gy$: internal bleeding, death

When the dose is low, the cell can repair the damage caused by irradiation. Unfortunately, the repair mechanism may not work entirely correctly. Therefore after irradiation, the cell starts to work abnormally. This abnormal behavior can develop

into cancer or genetic disorders due to excessive or incomprehensive division of damaged cells. It is important to note that damage caused by low-level radiation does not differ from damage caused by foreign chemicals or autoimmune disorders. [16]

1.1.6 Radiation Protection

In the previous section 1.1.5 are mentioned the health risks when tissue is exposed to ionizing radiation. Radiation is beneficial in different fields, and it is essential to remove hazards associated with radiation. One of this thesis aims to create a simulated environment with a radiation source and sensors. The properties of attenuation and shielding of radiation are used for the correct detection of radiation that passes through various materials. In the following subsection are discussed techniques to eliminate radiation.

Distance

The radiation be subject to *inverse square law*. This law says that radiation flux is proportional to the square of the distance from the point source. The *inverse square law* is described by following equation.

$$I \propto \frac{1}{r^2} \tag{1.26}$$

The distance from measurement to point source in the previous equation is denoted as r. This law is applicable for all types of radiation in a vacuum.

X-rays and gamma rays suffer a little absorption and dispersion in the air and, therefore, are suitable in the real world. Besides, the radiation consisted of huge particles such as α particles or electrons cannot be approximated by inverse square law. The massive particles are very reactive, and they lose their energy in the air very fast.

Shielding

The shielding is the most effective way how to decrease radiation exposure. When gamma or X rays pass through some material, they lose their intensity exponentially with distance. The following equation describes intensity behavior in the material.

$$I = I_0 e^{-\mu x} \tag{1.27}$$

I is the intensity of radiation (I_0 is the initial intensity), and x is the thickness of material in cm, and μ is the *attenuation coefficient* of the material with unit cm^{-1}. The *attenuation coefficient* depends on the material as well as on the energy of the ionizing radiation.

1.2 ROS and ROS2

Robot Operating System (ROS) [19] is a free, open-source framework that is used in the development of robotic systems. Many different approaches commonly used for robotic systems include various components, such as sensors, control software, data visualization, or actuators are provided. ROS *middleware* mediates communication infrastructure between sensors and software and provides hardware abstraction.

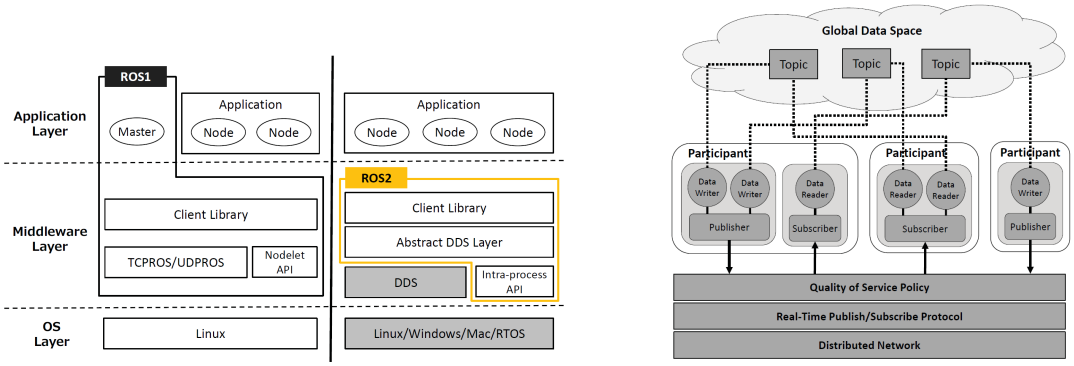
The ROS1 was developed by Willow Garage [23], and Open Robotics [24] from scratch in 2007. In recent years the amount of the ROS1 packages has grown exponentially [22]. "*Today we see ROS used ... on wheeled robots of all sizes, legged humanoids, industrial arms, outdoor ground vehicles (including self-driving cars), aerial vehicles, surface vehicles, and more.*[25]" The ROS1 was originally developed for the academic research community, but the ROS1 expanded beyond the academic sphere to a commercial area. [25]

ROS application is consists of independent *nodes* that contain program code that performs the computing process. The communication model between two or more *nodes* is *publish/subscribe* model.[28] Based on this model, the nodes exchange data to each other via *messages* which are clustered to *topics*. The *messages* are simple data structures that carry specialized information, such as odometry, sensors data, images, and more. The nodes that receive (subscribe) some *messages* are named a *subscriber*, and the node that sends (publish) *messages* is a *publisher*. The programming language used in ROS1 and ROS2[7] is C++ and Python, and all ROS1 environments run on Ubuntu OS. [19] [21]

A communication system in ROS1 is based on TCP and UDP sockets. Optionally, ROS1 provides *nodelet* [27] to use optimized, *intra-process* data transport. In ROS1, a *master* whose function is similar to a server function collects and manages information about all exchanged *topics*. Before starting to exchange data, *nodes* (*subscriber* and *publisher*) must interact (*XML/Remote Procedure Call*) with the *master*. After this procedure is enabled to move *messages* between *nodes* (*peer-to-peer*). ROS1 is not able to fulfill *real-time* requirements and cannot guarantee a *fault-tolerance* process. [28]

ROS2 is an improved and redesigned version of ROS1 which preserves time-proven methods and adds new use cases. It works above more commonly used operating systems such as Linux, Windows, and macOS. Unlike ROS1, ROS2 support *real-time* application and therefore support the *real-time OS*. Both systems are similar from the user's point of view. ROS2 also include *nodes*, *topic*, and *messages*, but it mainly differs in the *middleware* layer, as the picture below shows 1.5a.

[7] improvement of ROS1



(a) ROS1 and ROS2 architecture.

(b) Data-centric publish-subscribe model

Fig. 1.5: In the left figure is shown the architecture of ROS1 and ROS2 in terms of their layers. The figure on the right is the detail of the Abstract DDS Layer of ROS2 middleware. [28]

An aforementioned communication system in ROS1 is based on custom protocols TCPROS. For ROS2 is used existing *middleware* solution *Data Distribution Service* (DDS). There are different implementations of DDS that each has their own pros and cons (supported platform, programming language, performance, memory usage, and more). Therefore ROS2 supports many different implementations of DDS, and each implementation has a specific API. ROS2 provides *Abstract DDS Layer* that defines an interface between ROS and specific implementation.[26]

The data transfer in *DDS* is based on *Data-Centric Publish-Subscribe* model (DCPS). This model creates a global data space, and all *participants* can access it. *Participants* are called processes in DDS that can publish or subscribe data, same as in the ROS *node*. [28]

The DCPS model is shown in Figure 1.5b.

- *Publisher*: Is responsible for data issuance
- *Subscriber*: Is responsible for receiving data and making it available
- *Data Writer*: Must publish data through the *publisher*
- *Data Reader*: Receive, access data for the *subscriber*

Each data transmission is subject to the *Quality of Services* (QoS) policy such as *History*, *Depth*, *Reliability*, and *Durability* [28].

1.3 Robotic Simulators

Shannon defined the simulation in 1975 as "*the process of designing a model of a real system and conducting experiments with this model for the purpose either of*

understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system". [29]

Simulation plays an important role in robotics research. Nowadays, it is used for design, development, verification, validation, testing, and proving complex robotics systems' theoretical methods. Todays's robotics systems are consist of complex hardware devices and numerous sensors. Especially mobile robots are used in different tasks in various environments with multiple and changing conditions. All of these aspects make the development and testing of robots very expensive. The simulation allows developers or groups of developers to work with various robots or sensors without the risk of damage. Using well-developed simulations such as environment, sensors, actuators, and others reduces developing and testing time and makes it much easier and safer. [30] [1]

A robotics simulator is defined according to Jack Collins, Shelvin Chand et al. [31] as end-user software that includes the following features.

1. Physics engine
2. Collision detection and friction models
3. Graphical User Interface
4. Possibility to import scenes and meshes
5. API for programming language
6. Models of joints, actuators, and sensors

The 2020 survey [32] showed that approximately 72% of organizations use robotic simulators for their activities, which is shown in the figure 1.6. Furthermore, types of robotic simulators and their popularity which are commonly used [31] are shown in the figure 1.7.

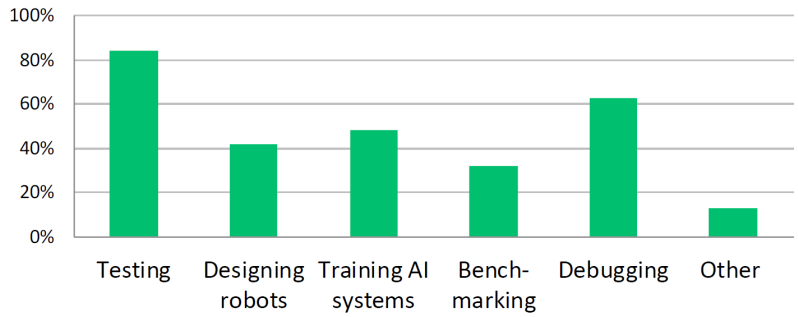


Fig. 1.6: A reason that participant organizations gave for using robotic simulation. [32]

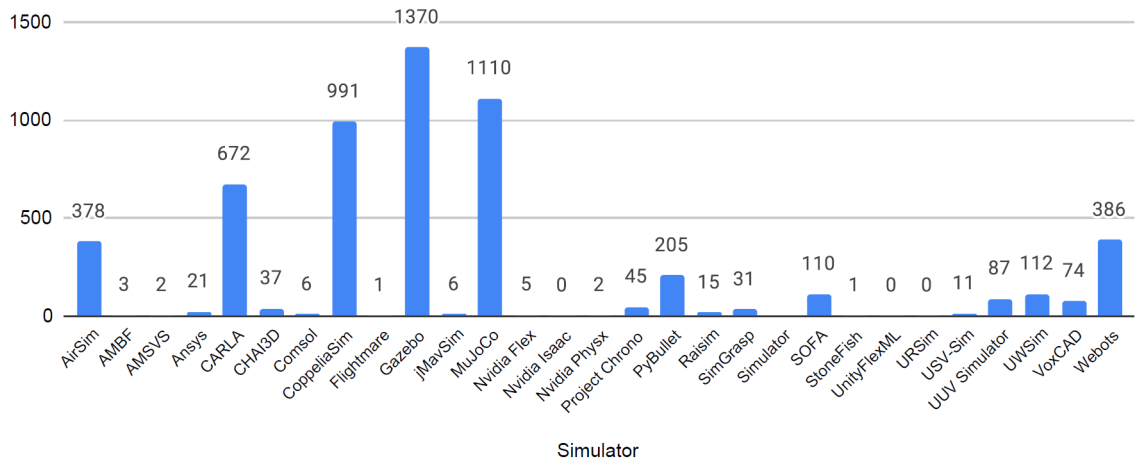


Fig. 1.7: Types of commonly used robotic simulators according to a number of citations on Google Scholar between 2016 to 2020. [31]

1.3.1 Robotic Simulators Overview

The Robotic Simulators sections described what a robotic simulator is and what types of simulators are used. The last figure 1.7 mentions the citation counts of reviewed simulators in the range of years 2016-2020. The most commonly used are *Gazebo*, *MuJoCo*, *CoppeliaSim*, *CARLA*, *Webots*, and *AirSim* descending in order. This chapter describes the suitability of using these simulators according to the diverse robotics areas.

Mobile Ground Robotics

One of the most extensively studied domains in robotics is mobile ground vehicles. The most popular simulator is *Gazebo*, and it is famous for the simulation of mobile ground robots. The *Gazebo* can cover all types of mobile robots mentioned earlier. The Robot Operating System simplifies creating control software and contributes to Gazebo's popularity. As well as the next rigid body simulator *CoppeliaSim* [34] (formerly called V-Rep), the *Gazebo* provides many commonly used sensors and allows to simulates multiple robots in real-time. Another popular alternative is *Webots* [37]. [31]

The specialized simulator of the mobile ground robot is *CARLA* [36], which focuses on simulation in a self-driving car, and its features are aligned with this goal. [31]

Manipulators

All of the aforementioned simulators except *CARLA* are suitable for creating robot manipulators. The manipulation in robotics is a large and diverse field. It deals with the physical design of arms and grippers, algorithms for planning and control, etc. Therefore, the second most cited robotic simulator from the figure 1.7 is *MuJoCo* [35]. The simulator *MuJoCo* stands out in contact stability. [31]

Aerial Robotics

Unmanned aerial vehicles (AUVs) are the most popular field in aerial robotics research. The UAV simulators simulate the complex real-world environment, including turbulence, air density, clouds, and other fluid properties. The last of a mentioned group of six is Microsoft's *AirSim* [33]. It is based on an Unreal engine and supports many UAV sensors, i.e., magnetometers, barometers, GPS, etc. The disadvantage of *AirSim* is computing power consumption against other simulators. [31]

Gazebo, *Webots*, and *CoppeliaSim* are also suitable to simulate aerodynamic properties. In *Gazebo* is implemented physical phenomena lift and drag in *LiftDrugPlugin* [6] and also support hardware controllers and Ardupilot and PX4 [31].

1.3.2 Gazebo

Gazebo[8] is an open-source robotic simulator developed by Open Source Robotics Foundation[9]. Well-designed robotic simulators accelerate the development time primarily test developed algorithms, train and testing Artificial Intelligent systems. One of the most significant advantages of simulator usage is that the Gazebo provides realistic scenarios based on a real physical engine. [6]

The Gazebo has become the most popular robotic simulator (see 1.3.1) because it provides many options to simulate indoor and outdoor three dimensions environments. It also simulates many sensors which are protected before the damage unlike the real-world experiments. It brings a wide variety of robots, objects, and simulation models[10] due to the large Gazebo community.

Gazebo application is split into two main parts. It is divided into *gzserver* which executes the physics update-loop and sensor data generation, and *gzclient* which runs graphical users interface with the possibility of setting and controls simulation properties and run other applications embedded in a Gazebo. *Gzserver* is independent on *gzclient* and can be launched without *gzclient*. It also supports remote launch on cloud computers without user interfaces. *Gazebo*

[8] `http://gazebosim.org`

[9] `https://www.openrobotics.org`

[10] `https://app.ignitionrobotics.org/fuel/models`

Besides *gzclient* and *gzserver* separation is the Gazebo architecture distributed between separate libraries. [6]

- Gazebo Master
- Communication Library
- Physics Library
- Rendering Library
- Sensor Generation
- GUI
- Plugins

Gazebo handles four Physics Library in Gazebo Master. Physics Library provides simulation of components, including rigid bodies, collision shapes, joints, and their constraints, frictions, slipping. Each Physics Library (Open Dynamic Engine - ODE[11], Bullet[12], Simbody[13], Dynamic Animation and Robotics Toolkit - DART[14])has its own properties and can be set in World Files.

Another division of the Gazebo components does not deal with internal structure but is essential for using the Gazebo simulator. When the Gazebo starts, it can be attached World File (or implicitly is load world) defines the properties and object layout.´Models can be spawned from Models Files to world loaded by the Gazebo. Properties or behavior can be affected by Plugins.

World Files

World files describe all simulation elements such as models, robots, objects, light, gravity, wind, etc. This file has .world extension, and a file is described in XML Simulation Description Format, SDFormat, or SDF [38].

The following code shows the basic structure of the world file.

```
<?xml version='1.0'?>
<sdf version='1.9'>
  <world name='default'>
    <physics type="ode"> ... </physics>

    <scene> ... </scene>

    <model name="box"> ... </model>

    <model name="sphere"> ... </model>
```

[11] http://ode.org
[12] https://pybullet.org/wordpress
[13] https://simtk.org/projects/simbody
[14] http://dartsim.github.io

```
    <light name="spotlight"> ... </light>

    <spherical_coordinates> ... </spherical_coordinates>
  </world>
</sdf>
```

The code, as mentioned earlier, is self-explanatory, and the complete documentation can be found on SDFormat website[15]. This SDFormat is used for the description of Model Files 1.3.2

Description complex indoor environment consisting of many walls, doors, and objects can be tedious and error challenging. In the Gazebo's graphical user interfaces is inbuild functionality called *Building Editor* with the help of which can be created complex indoor multi-floors building.

Model Files

Model files are like world files with the difference in SDF files must be only one following element. It is recommended to reuse the code. [6]

```
<model name="default">
  ...
</model>
```

As the building editor in a complex world file generation is a build-in functionality for model creating called the Model Editor, which is launched from the Gazebo.

Every model in SDFormat is consisted of Links, Joints, and Plugins according to the following structure.

```
  <model name="box">
    <pose>x y z r p y</pose>
    <static>false</static>

    <link name="link">
         <inertial> ... </inertial>

        <collision name="my_collision"> ... </collision>

        <visual name="my_visual"> ... </visual>
    </link>
```

[15] http://sdformat.org/spec

```
    <joint type="revolute" name="my_joint"> ... </joint>

    <plugin filename="libMyPlugin.so" name="my_plugin"/>

  </model>
```

Inertia element describes dynamic properties, mass, and rotational inertia matrix. Every link has its own Collision and Visual properties. These properties are separated due to maintaining simplicity for the physical library and saving the computation power.

The SDF model file is significantly longer with increasing robot complexity because it is essential to write similar code for similar links. For example, four links for four wheels of the robot. The Unified Robot Description Format (URDF) [39] is also an XML format used in ROS, allowing adding parameters to XML files, reusing code with XACRO (XML macro) format, and reducing the number of line lines in the description file.

Plugin

Plugins provide a convenient mechanism to affect and control almost arbitrary behavior of Gazebo and simulated objects. For affecting different components of the Gazebo, there are six different types of Gazebo plugins.

1. World
2. Model
3. Sensor
4. System
5. Visual
6. GUI

The world, model, and sensor plugins are attached and affect (provide some functionality) relevant parts of the object.[6] For example, the world plugin can add wind, change the atmosphere, light properties, etc. Often used is the Sensor plugin, which can inherit from different types of sensors and change their properties or use similarities and create a new type of sensors.

1.3.3 Ignition Gazebo

"Simulate before you build"[40] is the motto of Ignition Robotics. Ignition Gazebo, part of the Ignition Robotics project, is the same as the Gazebo open-source robotic simulator, which brings a fresh approach to simulation with a complex toolbox, libraries, and cloud services. Ignition will be the replacement from the Gazebo

simulator, and it's directly derived from it. Ignition Gazebo is focused on high-performance dynamics simulation based on Ignition Physics[16], advanced 3D graphics, sensors, and noise models, plugins, communication, etc. [40]

The Ignition Gazebo is encompassed from Ignitions Libraries. The simulator is divided into two independent launched processes. A backend server is responsible for the "simulation loop" which computes physics, recording logs, receiving commands, etc. A frontend process primarily stands on the Graphical User Interface (GUI) and rendering scene. Communication between the backend and frontend server is based on Ignition Transport library[17]. Ignition Gazebo is consists of several plugins (libraries). The plugins are modular. Therefore, these plugins are independent of each other components and can be loaded at runtime with some exceptions. One of the exceptions is Ignition Common[18] which provides common functionality to all plugins. Some of the plugins run default when starting the Ignition Gazebo, but all are optional, and the user can remove it or add other plugins. [40]

It is essential to communicate between ROS (both ROS or ROS2) and the simulator to develop the robotic system. Similarly, as gazebo_ros_pkgs[19] in the Gazebo, the ros_ign_bridge[20] package enables the exchange of messages between ROS and Ignition Transport.

[16] https://github.com/ignitionrobotics/ign-physics
[17] https://ignitionrobotics.org/libs/transport
[18] https://ignitionrobotics.org/libs/common
[19] https://github.com/ros-simulation/gazebo_ros_pkgs
[20] https://github.com/ignitionrobotics/ros_ign

2 Environment

This thesis has proposed some approaches to creating a world environment essential for good approximation for future real-world experiments. The simulated world can be made from artificial solid objects, laser scans or grayscale images, DEMs, and 3D point clouds. This section provides an overview and performance evaluation of the simulated world.

The performance[1] and CPU load of the Gazebo affects two factors with different properties. A face's - the surface between three or more edges - number and number of robots. The more restrictive factor on the Gazebo performance is the number of robots in multi-robot scenarios depicted in the following experiment. This experiment was performed with hardware: Intel(R) Core(TM) i7-5500U CPU @2.4 GHz 2 Cores (4 Logical) 8 GB RAM Intel(R) HD Graphics 5500 NVIDIA GeForce 940M and software: Gazebo 11, ROS2 Foxy, Ubuntu 20.04 LTS. Into the Gazebo was gradually spawned from zero to four Orpheus X4 robot to the empty world - default world - and recorded the real-time factor (RTF), CPU load, and memory usage of gzserver. RTF is a ratio of simulation time to real-time.

The following diagram shows the rapid RTF decreasing according to the number of robots in simulation. This constraining allows spawning to the simulation only a small number of movable robots (in this case is Orpheus X4).

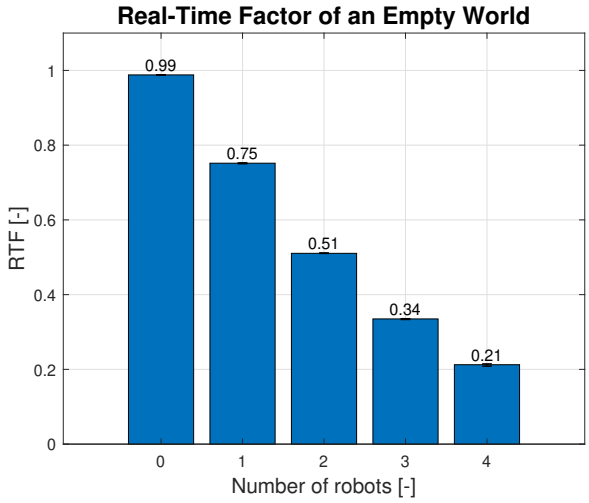
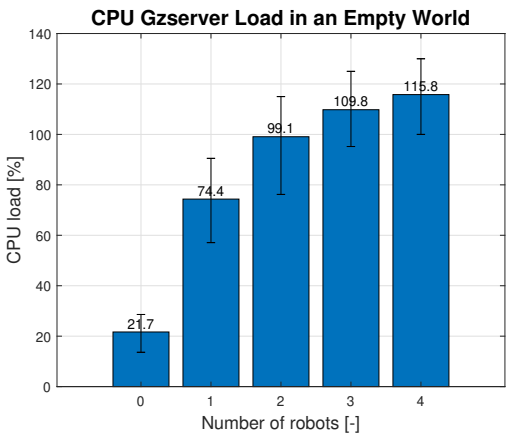


Fig. 2.1: The Gazebo simulator's Real-Time Factor (RTF) depends on the number of spawned robots in the simulation.

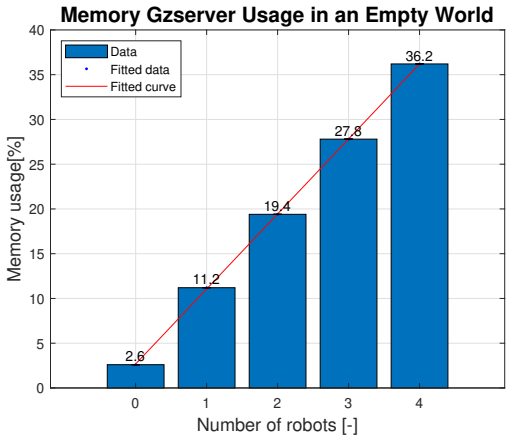
The CPU load and memory usage of gzserver were recorded after spawning the robot into the simulation. The relationship between CPU usage and the number of

[1]Performance of the Gazebo is RTF in this case

robots in the simulator represents the graph 2.2a. CPU load doesn't grow so fast as the memory usage of gzserver. More robots were not spawned because RTF dropped very low. Dropped RTF is why CPU loads were gone to the final value and did not consume more computer power.



(a) CPU load of gzserver depending on number of robots



(b) Memory usage of gzserver depending on number of robots

Fig. 2.2: On the left figure is the CPU load of gzserver and the right's picture shows the memory usage depending on the number of robots in the simulation

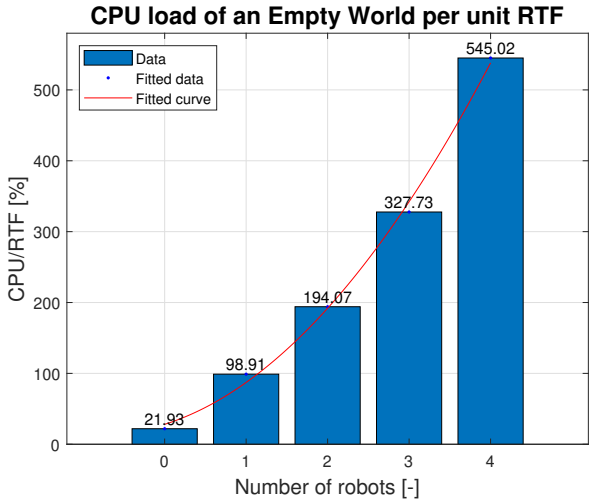


Fig. 2.3: The evaluation of the Gazebo simulator according the CPU load per unit RTF. The data represents the estimation CPU load (100% = 100% use of one core) when RTF equals one

The memory usage of gzserver has the linear character that is described with an equation $M(k) = 2.68 + 8.38k$, where k (non-negative integer) is the number of

robots, and $M(k)$ is memory usage in percentage. Every next robot consumes 8.38% of memory. The graphs 2.2 show the aforementioned relationship.

The previous RTF graph 2.1 or RTF value in the Gazebo provides useful and fast knowledge about the Gazebo performance, but it does not provide a reliable relationship between computing requirements and the number of simulated robots, same as CPU or memory usage. The following graph 2.3 shows the ratio of CPU load to RTF. This dependency describes theoretical computing requirements for simulation where simulation time is equal to real-time (RTF = 1).

The graph 2.3 shows that the computing requirements increase quadratically depending on the number of simulated robots with the extrapolating equation $C(k) = 22.79k^2 + 36.33k + 28.12$. $C(k)$ is the estimation of the computing power of one core in percentage. k (non-negative integer) is the number of robots.

2.1 Environment from Model Database

The file that describes a world has a .world extension, as the previous section Gazebo describes. One of the ways to create an environment is to define or include the static object inside the <model> ... </model> structure. The inserted object can be acquired from the gazebo_models database [41], ignition database [2], or create its own object in the Blender[42] or any other 3D computer graphics software. This way is tedious and error challenging for a large number of inserted objects. A faster approach with a less clear .world file is to use the drag and drop mechanism of the Gazebo's GUI. The following figure 2.4 shows an example of a thus created outdoor environment with collapsed factory after some disaster for a robot rescue mission. This environment had formed from the gazebo_model database [41].

Fig. 2.4: The example of the world created from an object from the gazebo_models database.

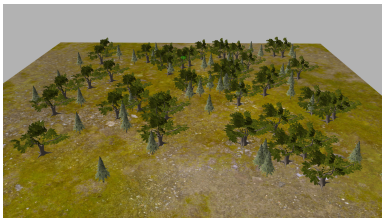
[2] `https://app.ignitionrobotics.org/fuel/models`

2.2 Generation of Random Environment

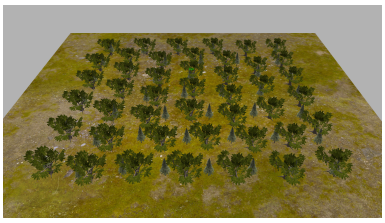
One of the most significant advantages of robotic simulators is the possibility of creating a large number of different worlds and fast testing developed algorithms. Creating an environment from the model database is a suitable solution for creating one complex environment. Nevertheless, pathfinding, collision avoidance, SLAM, search and rescue, and other robotics tasks need evaluation and verification according to the varied environment.

Population of Models

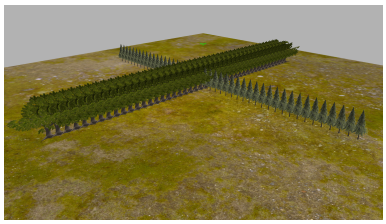
The Gazebo simulator offers an excellent feature[3] for inserting a population of identical models into the world. The population of models is defined in <population> ... </population> tag in SDF format. Population tag includes information about the position and orientation of the area in which the models are spawned, number, and type of models in the defined area. This feature allows defining different properties of the distribution model in the area, as the followings graphs show 2.5.



(a) Random distribution



(b) Grid distribution



(c) Linear-x and linear-y distribution

Fig. 2.5: Overview of distribution option in the population of models

The population model can be used for inserting requisition subject to the simulated world. For example, in this thesis, the population of models can randomly add lost radiation sources[4] to the world already created.

[3] `http://gazebosim.org/tutorials?tut=model_population&cat=build_world`

[4] or another point of interest

2.3 Building Editor

The Gazebo offers an inbuild application called building editor[5] for robotic tasks inside the building. The building editor has a simple GUI divided into three parts. In the first part, the type of component is selected to build. It offers walls, doors, windows, stairs, and color and texture. The building is designed from this component in a 2D window. In the 3D window, under the 2D window, is visualized model from the 2D drawing editor. It is possible to design buildings with the floor or import floor plans. The Gazebo building editor crashes when adding a door or windows in the 2D editor for hardware and software configuration listed in previous chapters. The other users of the Gazebo building editor have the same issue that is solved on the official Gazebo GitHub page[6].

2.4 Environment from Real Data

Creating an environment similar to real-world, robot models with proper physical properties, and approaching of tests results in simulation and real-world is the aim of the robotic simulator. The use of software without whatever changes in simulation and real-world leads to faster and cheaper development. In these assumptions, the natural idea is to use real data to set up a simulation, especially to create an environment (world). Such an environment/world allows repeating the exact same experiment in real-world. Comparing the results can find the differences and improve the simulation or allow more to rely on simulation only. This section suggests some options for creating/modeling the world based on real data.

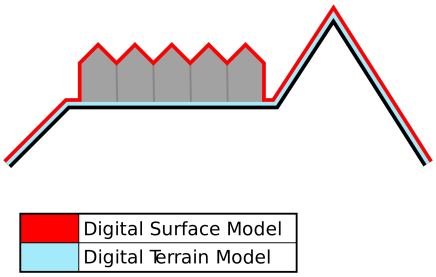


Fig. 2.6: The difference between Digital Terrarian Model (DTM) and Digital Surface Model (DSM) [43]

All of the worlds/environment above was built on a plane surface. For outdoor UVG is appropriate to build a world with the terrain. One of the options is model's

[5] http://gazebosim.org/tutorials?tut=building_editor&cat=build_world
[6] https://github.com/osrf/gazebo/issues/2276

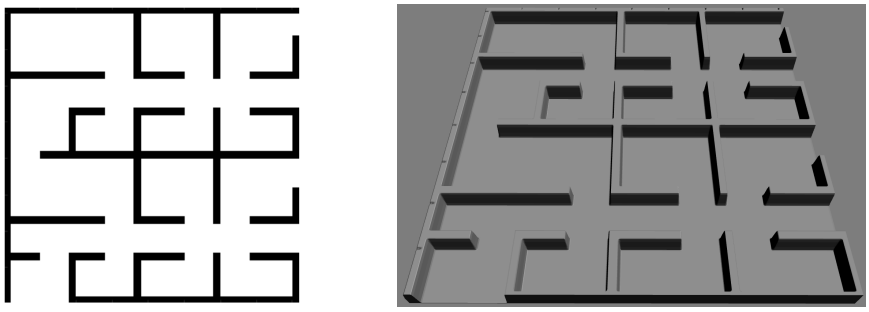
based. The surface can be modeled in 3D modeling software and imported as a static object to the world file. The next option is to use *Digital Elevation Model* (DEM). This DEM embodied the surfaces elevation of the Earth, the Moon, or other celestial bodies. DEM is suitable for a large-scale outdoor environment. For simulation can be used two types of DEM. *Digital Surface Model* (DSM) or *Digital Terrain Model* (DTM). Differences between DSM and DTM are shown in the picture 2.6.

Point cloud as a method for representing data in 3D space is the next option to create a real environment. Model created from the point cloud is loaded to Gazebo as mesh COLLADA[7] file (with extension .dae). This method can be utilized for indoor and outdoor applications.

The following worlds are generated from data obtained from the mission described in An automated heterogeneous robotic system for radiation surveys: Design and field testing [44].

2.4.1 Digital Elevation Model

There are several ways to transform DEM into an importable model for the Gazebo. DEM can be represented as a grayscale image that is needed for the automatic tool (LIRS World Construction Tool or LIRS-WCT [3]) for Gazebo world construction developed by Bulat Abbyasov, Roman Lavrenov et al. This tool generate Collada format from a grayscale image that could be directly imported into the Gazebo simulator [3]. LIRS-WCT tools can be used for creating the terrain model or reconstruction 2D occupancy grid (or simply handmade occupancy grid), as shown in the following example 2.7.



(a) The grayscale pattern for LIRS-WCT

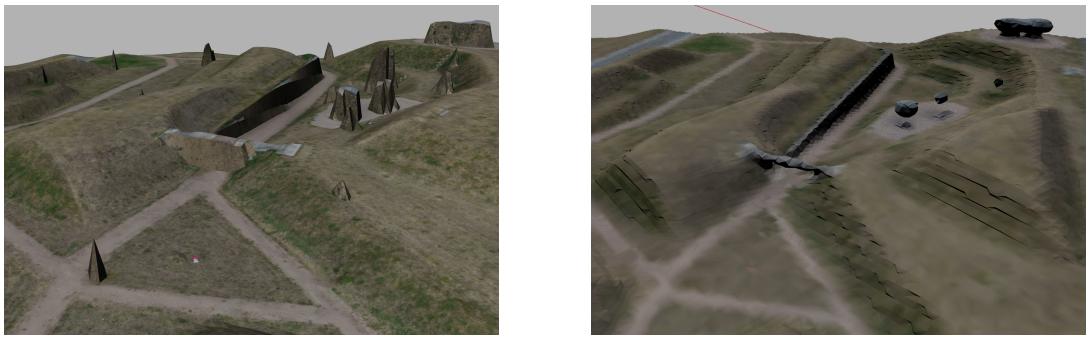
(b) Solid Collada model output from LIRS-WCT

Fig. 2.7: Process of creating a model for the Gazebo simulator using LIRS-WCT. An occupancy map on the left where a black pixel represents occupied cells and white free cells. An output model is shown in the Gazebo.

[7] based on XML same as SDF

Another tool (Gazebo_heightmap_preparation) to transfer occupancy maps to the Gazebo world is described in the article: "Tool for 3D Gazebo map construction from arbitrary images and laser scans [2]". LIRS-WCT has significantly better performance than Gazebo_heightmap_preparation [3].

Following tools can be used to create a 3D terrain model is with the help of QGIS[45] software and plugin DEMto3D or using the Gazebo's[8] <heightmap> ... </heightmap> tag with GDAL libraries[9].



(a) The world create from DEM

(b) The world created from point cloud

Fig. 2.8: On the picture is shown two types of environment. On the top is a model from DEM and on the lower picture is a world created from a point cloud

In this example, the model in the picture 2.8a has many artifacts (cone or impassable bridge) that non correspond with reality caused by trees or climbing walls. This problem solves creating a model from a point cloud.

2.4.2 Point Cloud

This section describes the process of making a model/mesh from the point cloud consisting of at least two steps. For these steps are used CloudCompare[46], Meshlab [47], and/or Blender[42]. Before making a mesh (one step) is necessary to preprocess data from the point cloud. After preprocessing and creating a mesh is optional to extract and apply texture.

Point cloud preparation

A point cloud has a commonly wide range of points number depending primarily on the size of the displayed object. The 3D scan can consist of millions to hundreds of millions of points. Such point cloud is too dense, and therefore, it would need large

[8] http://gazebosim.org/tutorials?tut=dem&cat=build_world

[9] https://gdal.org/

computing, memory, and time resources. It is appropriate to reduce the point cloud with these assumptions.

Original point cloud has over 29000000 points spaced on the area with size $200m \times 170m$, approximately. To view and adjust point clouds is using a great tool, CloudCompare[46]. With this tool, point cloud reduction is performed with the inbuild Subsample[10] function. The Sumbsample creates a new subsampled/reducted point cloud, a subset of the original data. Subsampling can be performed with random, spatial, and octree[11] methods. The random method defines a final number of points, the spatial defines the minimum distance between two points, and the last octree method selects the level of subdivisions. For this thesis, the spatial method was selected with easily imaginable results. The value was set to 0.1 (CloudCopare does not use units. Units are inherited from an original point cloud. It is called implicit units), and the number of points was reduced to 2000000 approximately. Next point reduction can be made with filters[12][13] or manually remove visible artifacts, for example, trees without tree trunks. The point reduction is shown in the following figure 2.9.

(a) The world create from DEM

(b) The world created from point cloud

Fig. 2.9: An example of point cloud reduction. The left figure embodied the original point cloud with over 29 million points, and on the right is a reduced point cloud with the same point size.

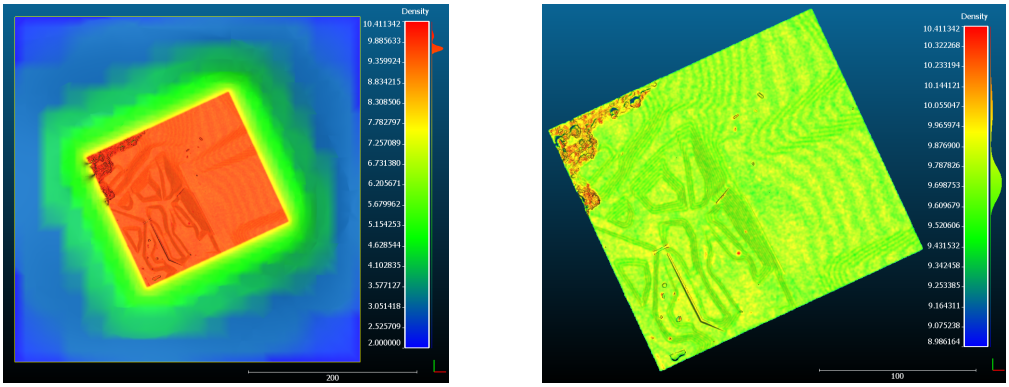
After reduction and/or filtering, a point cloud is about 5% to 10% of the original point cloud. Nevertheless, these points do not have any information about a local surface. The substep in point cloud preparation is computing normals that estimate a local surface and orientation represented by points and their neighbors. The options in tool Compute normals[14] are the neighbors and orientation. For the

[10] https://www.cloudcompare.org/doc/wiki/index.php?title=Edit%5CSubsample
[11] https://en.wikipedia.org/wiki/Octree
[12] https://www.cloudcompare.org/doc/wiki/index.php?title=Noise_filter
[13] https://www.cloudcompare.org/doc/wiki/index.php?title=SOR_filter
[14] https://www.cloudcompare.org/doc/wiki/index.php?title=Normals%5CCompute

neighbor is selected its number. The more points used for computing normals generate smoother and more consistent normals. Calculation time is, therefore, naturally longer. Orientation of normals can be computed with Minimum Spanning Tree[15] or can be preferred certain orientation (this example, it is +Z axis).

Create a Mesh

After the normals are computed, it is possible to create a mesh model. The normals must be clear and consistent for a good mesh reconstruction from the point cloud. This step uses the plugin PoissonRecon, commonly used for closed 3D shapes, and therefore, the PoissonRecon extends the original output mesh. Output density as SF[16] (scalar function) can be easier used to remove this redundant mesh from the model on picture 2.10a. The sparse spatial density is shown with blue color. For removing this redundant mesh is used tool Filter by Value[17]. Boundary values for this function is acquired just from the scalar function 2.10.



(a) Before filter by scalar value

(b) After filter by scalar value

Fig. 2.10

[15] https://www.cloudcompare.org/doc/wiki/index.php?title=Normals%5COrient_Normals_With_Minimum_Spanning_Tree

[16] https://www.cloudcompare.org/doc/wiki/index.php?title=Poisson_Surface_Reconstruction_(plugin)

[17] https://www.cloudcompare.org/doc/wiki/index.php?title=Scalar_fields%5CFilter_by_Value

3 Robot Models

In previous chapters 2, some options were introduced on how to build and simulate robot environments. Both outdoor and indoor environments were mentioned, as well as the creating environment from real sensor data. They also outlined the Gazebo simulator performance statistics, limiting the number of simulated robots.

This chapter dealt with the mobile robot model used in this work for complete radiation search tasks. First, the creation of a physical robot model in the simulator is described. The kinematics and dynamics are described for such a model. Dissipative forces are discussed based on the physical properties affecting the model's behavior in the simulator. To complete comprehensive robotics tasks is necessary to combine many processes, ensuring robot localization, control robot motion, trajectory generation, path following, and more. For complete control motion, robot models are derived and compared with simulated data.

3.1 Orpheus X4

The motion of the mobile robots is a result of the interaction of a wheel and a ground. According to wheel slippage, mobile robots can be divided into two types. [52]

1. Robots do not occur sliding (or minimalizing sliding) during a motion
2. Robots it's a feature of the movement is sliding

Robots belonging to the first category typically work in indoor environments with a flat floor. The second category usually includes more than two-wheeled and non-steered robots that can move in rough outdoor terrain. The four wheels mobile robot Orpheus X4 3.1 was chosen for radiation search tasks in this thesis. The four-wheeled mobile robot was selected for its mechanical robustness due to the lack of a steering system and good maneuverability.

Fig. 3.1: Four-wheel Orpheus X4 robot for work in inaccessible or dangerous areas (`http://cafr.cz/orpheus---x4.html`)

3.2 Robot Model in Gazebo

Simulators in robotics research have many benefits in the form of acceleration and cheaper development, easy to perform experiments, protecting physical damages, etc., as shown in the Robotic Simulators Overview chapter about simulators. This subsection 3.2 describes the process of making a robot model in the Gazebo simulator in more detail and will be mentioned different methods of describing the robot and some crucial physical properties of links and joints (basic building elements).

3.2.1 Methods for Defining Models

There are three main ways to build (define) and simulate robots or other models in the Gazebo simulator. The first fundamental approach is to define the robot model via *SDFormat* 1.3.2. Using SDFormat is suitable for defining elementary physical objects, including one or two links. Otherwise, the output file describing the robot is long and confusing because of the necessity to redefine similar links. For example, the wheel with the same physical properties must be defined four times instead of using macro and reusing the code. **U**nified **R**obot **D**escription **F**ormat[1] (*URDF*) allows using and clearing up of the description file with **X**ML m**acro** (*Xacro*). Reusing the code with *Xacro* is fulfilled with the following code structure.

```
<xacro:macro name="wheel" params="prefix...">
    ...
</xacro:macro>
```

Between these two tags are defined robots' links, for example, the wheel.

```
<xacro:wheel prefix="left_wheel" ...  />
<xacro:wheel prefix="right_wheel" ... />
```

The URDF description is inserted into the appropriate place from the macro by calling the macro.

```
<xacro:arg name="simulated" default="true" />
<xacro:property name="wheel_radius" value="0.19"/>
<xacro:property name="wheel_width" value="0.055"/>
```

XML also supports the variable. Defining constant in one place brings an obvious advantage.

URDF is also an XML file format, like SDF. However, proper simulation is needed to add some simulation-specific tags 3.2.2, defining the suitable physical properties for simulation, see next section. The last approach is to use drag and drop build-in

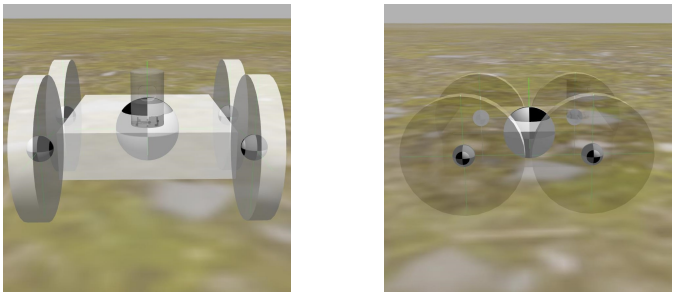
[1] `http://wiki.ros.org/urdf`

Gazebo function Model Editor, which generates 'unapplicable' code for the human, which is not recommended to modify manually.

3.2.2 Physical Properties

The object behavior is dependent on physical properties that can be managed in a few types of Physics Library (dart, simbody, bullet, ode) that can be chosen in the world SDF description file. Some general settings can be adjusted in the world SDF file, such as real-time factor and max step size. These settings affected the behavior of the method of calculating, but this chapter will be mentioned some interesting properties affecting mobile robot behavior directly.

The URDF is the best practice to define robot properties. For accurate simulation, the URDF file must fulfill one requirement. Each robot's link must contain an inertia tag. Together with collision properties, the inertia tags (all robot links) describe the robot's dimensions, mass, weight distribution 3.2, and moment of inertia tensor. All of these properties can be easily gained from an actual robot structure, and they do not change during the experiments in most cases.



(a) Front view (b) Side view

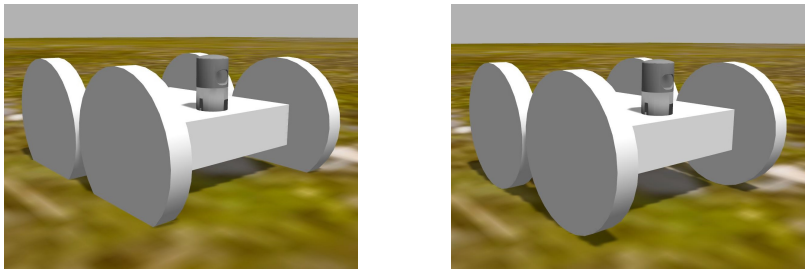
Fig. 3.2: The robot's center of mass

In addition to basic physical properties, the Gazebo allows defining optional parameters in <gazebo> tags for every joint, link, and whole model. One of the most attractive properties[2] are mentioned in the following division.

- Model
  - static - Sets the mobility and immobility of the object
- Links
  - maxVel - maximum correction velocity when contacts
  - minDepth - minimum depth before maxVel is applied
  - mu1 - friction coefficient in fdir1 direction
  - mu2 - friction coefficient iin the direction perpendicular to fdir1

[2] `http://gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros`

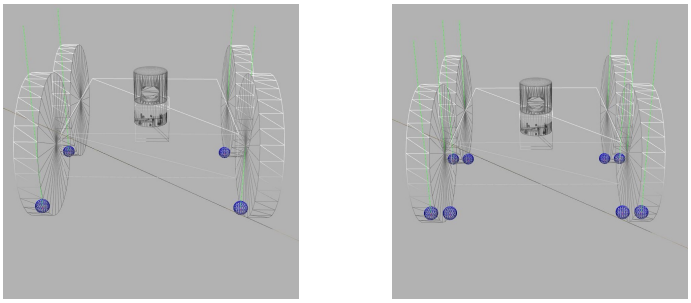
  - fdir1 - direction
  - maxContacts - maximum allowable number of contacts between two links
- Joints
  - dynamics
    - friction - define static friction force of joint in N or Nm
    - damping - define damping force in Ns/rad or Nms/rad
  - limit



(a) minDepth = 0.1m (b) minDepth = 0.001m

Fig. 3.3: Parameter minDepth and its visualization in Gazebo

The properties maxVel and minDepth 3.3 work together and affect the robot's or model's bounciness and stickiness. Values are not intuitive and determinable, unlike weights, etc. Their setting is an iterative process. It is appropriate to set the default[3] value on minDepth to $0.002m$ and maxVel to $10ms^{-1}$. After performing the experiment and evaluating the model's behavior is suitable to change the minDepth and maxVel values. The following graph 3.5 shows the experiment's result when the robot was spawned to Gazebo from one meter above the ground at different values of minDepth and constant value of maxVel. Increasing minDepth is increasing stickiness and decreasing bounciness.



(a) maxContacts = 1 (b) maxContacts > 1

Fig. 3.4: Parameter maxContacts and its visualization in Gazebo

[3]Default value for settings in this thesis. This does not default values in Gazebo

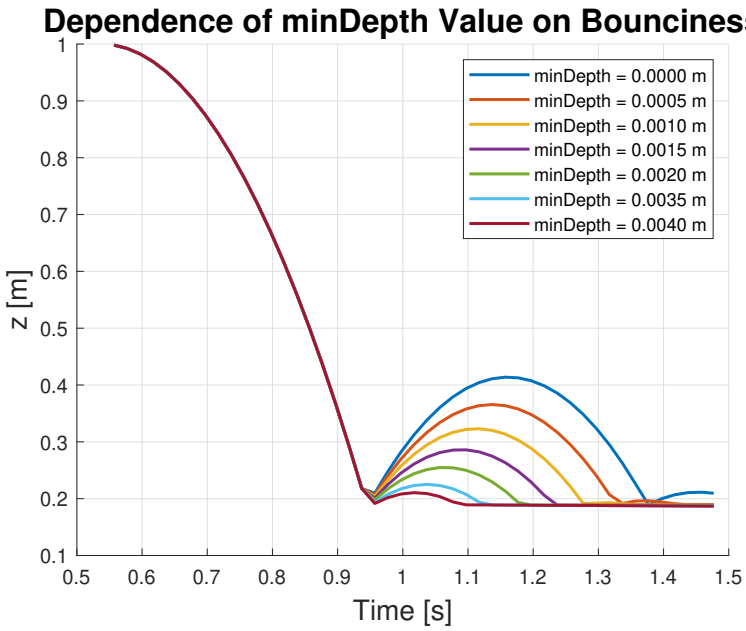


Fig. 3.5: The influence minDepht parameter on bounciness of spawned robot. The graph shows the height of robots depending on the time with different values of minDepth.

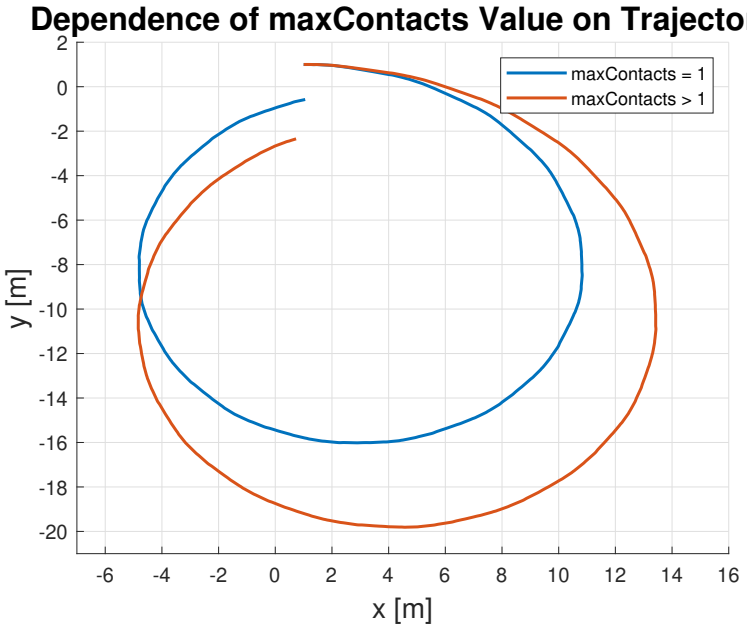


Fig. 3.6: The wheels have only one contact between the wheel and the ground in the left picture. The position of contact oscillates from one side of the wheel to another. The robot has two contacts simultaneously, on the right image.

The maxContacts parameters play an important role in the contacts of complex objects, especially when a collision is a mesh object. In this thesis, the maxContacts parameter is set for the wheels. The picture 3.4 shows two cases setting of maxContacts parameter. The robot on the left image has set maxContatcs to one, and on the right, maxContacts to five. There are a maximum of two contacts with the ground so that the parameter can be set greater than 1 with the same result, in this case. Because the wheel of the robot has the shape of an ordinary cylinder. The robot's behavior is shown on the graph 3.6 with different settings of maxContacts.

3.3 Motion Analysis

Knowing robot kinematics is essential to understanding how the mechanical system behaves. And kinematics is necessary to design control software also. One of the features of knowing how robots move is the possibility to estimate the motion. Motion estimation is an extremely challenging robotics task due to the necessity to integrate wheels velocity loaded by inaccuracy. Dead reckoning, one of the methods to determine a robot's position, is not suitable for four-wheeled robots, where slippage is the basic property of the robot. Nevertheless, the equation describes the four-wheel robot motion used to develop the software. The kinematics analysis is the well-described issue, and this description is based on this [52] [53] [54] [55] articles.

It is necessary to define some assumptions before describing kinematics equations.

1. Between ground and wheels are the only point contact
2. All wheels are always in connection with a surface
3. The wheels on the same side have the same rotation speed
4. The robot moves on a horizontal plane
5. The longitudinal slippage is neglected

Assumption 4. limits the movement only on the plane 3.7; therefore, the robot state can be described as a position ${}^{G}\vec{q}$ in the fixed global $X_gY_gZ_g$ coordinate system $\{G\}$.

$$ {}^{G}\vec{q} = ({}^{G}X \ {}^{G}Y \ {}^{G}\theta)^T \tag{3.1} $$

And as the robot velocity ${}^{G}\dot{\vec{q}}$ in the coordinate system G and robot velocity ${}^{R}\dot{\vec{q}}$ in the moving frame $x_l y_l z_l$ attached to the robot COM (Center of Mass) $\{R\}$.

$$ {}^{G}\dot{\vec{q}} = ({}^{G}\dot{X} \ {}^{G}\dot{Y} \ {}^{G}\dot{\theta})^T \tag{3.2} $$

$$ {}^{R}\dot{\vec{q}} = ({}^{R}\dot{x} \ {}^{R}\dot{y} \ {}^{R}\dot{\theta})^T \tag{3.3} $$

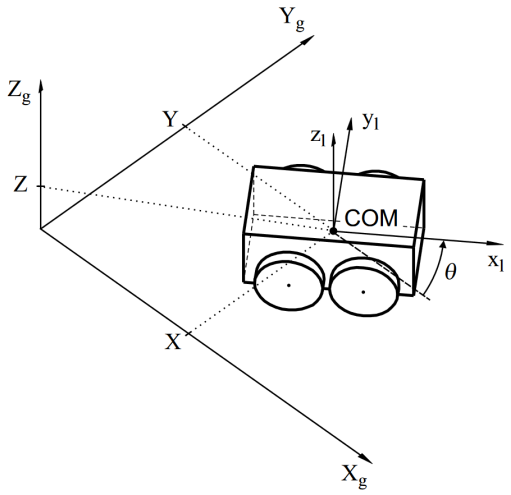


Fig. 3.7: Robot coordinates system of four-wheel mobile robot. [53]

Where $\dot{X}$ and $\dot{Y}$ are robot *COM* velocity in frame G (/odometry frame according to Coordinate System) and $\dot{\theta}$ is robot angular velocity according to Gz axis counterclockwise. See picture 3.7.

Velocities in coordination systems G and R are in the following relationship,

$$^{G}\dot{\vec{q}} = {}^{G}\mathbf{R}^{R}\,{}^{R}\dot{\vec{q}} \tag{3.4}$$

where rotation matrix ${}^{G}\mathbf{R}^{R}$ has the following form.

$${}^{G}\mathbf{R}^{R} = \begin{pmatrix} cos({}^{R}\dot{\theta}) & -sin({}^{R}\dot{\theta}) & 0 \\ sin({}^{R}\dot{\theta}) & cos({}^{R}\dot{\theta}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

The equation 3.5 allows unlimited robot velocity in a lateral ${}^{R}\dot{y}$ direction. Assuming that *COM* is the same as *COG* (Center of Geometry), the robot velocity ${}^{R}\dot{y}$ in the lateral direction can be considered zero. The equation 3.4 can be simplified to the following form.

$${}^{G}\dot{\vec{q}} = \begin{pmatrix} cos({}^{R}\dot{\theta}) & 0 \\ sin({}^{R}\dot{\theta}) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} {}^{R}\dot{x} \\ {}^{R}\dot{\theta} \end{pmatrix} \tag{3.6}$$

The motion of the mobile robot is based on an interaction between wheels and the ground. Therefore it is necessary to obtain robot velocity from the angular velocity of each wheel. From assumption 3) angular velocity, right and left wheels are the same, respectively. Let denote ω_r and ω_l as an angular wheel. Parameter t is the half span of the robot. From assumption 1. span is computed as the distance between the centers of wheels.

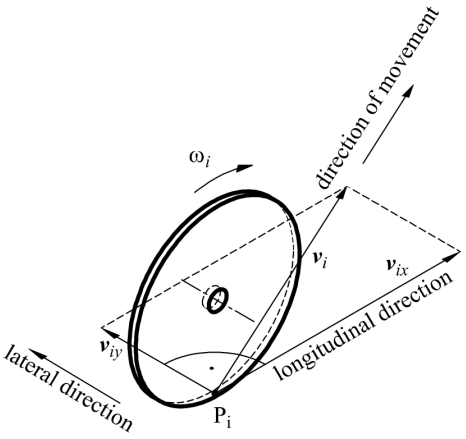


Fig. 3.8: Robot wheels' velocities [53]

The lateral velocities of wheels are generally nonzero 3.8 because the four-wheeled mobile robot's properties slip when changing its orientation. Wheels are tangent to the path only when ${}^{R}\dot{\theta} = 0$ [53]. For simplicity is, according to [53] and [56], are rotational velocities

$$\vec{\omega}_w = \begin{pmatrix} \omega_l \\ \omega_r \end{pmatrix} = \frac{1}{r} \begin{pmatrix} v_l \\ v_r \end{pmatrix} \tag{3.7}$$

Where v_l and v_r are left and right longitudinal[4] wheels velocities, respectively. Effective rolling radius is denoted as r. The relationship between wheel velocities and the robot's velocity is shown in those articles [53] and [54].

$$\begin{pmatrix} v_l \\ v_r \end{pmatrix} = \begin{pmatrix} 1 & -t \\ 1 & t \end{pmatrix} \begin{pmatrix} {}^{R}\dot{x} \\ {}^{R}\dot{\theta} \end{pmatrix} \tag{3.8}$$

From equations 3.7 and 3.8, the following mutual relationship between the rotational velocity $\vec{\omega}_w$ and the robot velocity in frame R is obtained.

$$\begin{pmatrix} {}^{R}\dot{x} \\ {}^{R}\dot{\theta} \end{pmatrix} = r \begin{pmatrix} \frac{\omega_r + \omega_l}{2} \\ \frac{\omega_r - \omega_l}{2c} \end{pmatrix} \tag{3.9}$$

3.4 Dynamic Model

A dynamic model of a skid steering mobile robot (*SSMR*) plays an essential role in control applications such as trajectory tracking problems. Sliding is an integral part of robot motion, especially in changing its orientation; therefore, the dynamic

[4]The longitudinal velocity of each wheel correspond with Rx axis

model highly affects the robot motion much more than the kinematics model and knowing about wheel rolling. One of the goals of a robotics simulator is to simulate a real system authentically. Knowledge of the model of the simulated robot is much easier to set appropriate properties in the Gazebo simulator concerning real-world experiments. Therefore in this section is derived the dynamic model of the four-wheel *SSMR* with respect to the physical properties of the Gazebo model described in the previous chapter. This section also discussed the resistive forces affecting robot motion and model verification with Gazebo experiments. It is necessary to define assumptions in advance at defining an equation of motion.

1. Motion is neglected at the y-axis.
2. Low movement speed (to $3ms^{-1}$ and $3rads^{-1}$)
3. Assumptions from Motion Analysis unless otherwise stated

The wheel and robot motion is based on control input torque τ_i, which is in following mutual relationship with forces F_i acting on each wheel, see picture 3.9.

$$F_i = \frac{\tau_i}{r} \tag{3.10}$$

According to assumption and minimalizing the longitudinal slippage, wheel torque τ is the same on both wheels on the left and the right side, respectively.

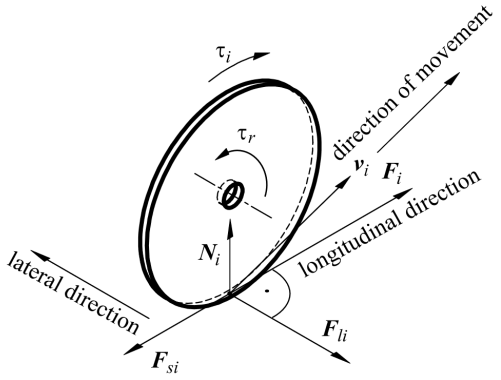


Fig. 3.9: Forces performing the wheel motion [53]

The equation of robot motion in a longitudinal direction can be expressed as the following equation, where the mass of the robot is denoted as m, forces causing the movement as F_l and F_r. Resistive forces F_{static}, $F_{dynamic}$, $F_{sliding}$ and $F_{rolling}$ are detailed described in section Dissipative Forces.

$$m\ddot{x} = 2F_l + 2F_r - F_{static} - F_{dynamic} - F_{sliding} - F_{rolling} \tag{3.11}$$

The acceleration is related to coordinate frame R, and the forces from the equation have the same orientation as the x_l axis on the picture 3.7. The model was

realized in Matlab Simulink. The following picture 3.10 shows the schema of longitudinal acceleration and velocity, respectively. On the bottom of the scheme are the outputs of right and left wheel resistive forces denoted in the future equation deriving as F_{RR} and F_{RL}.

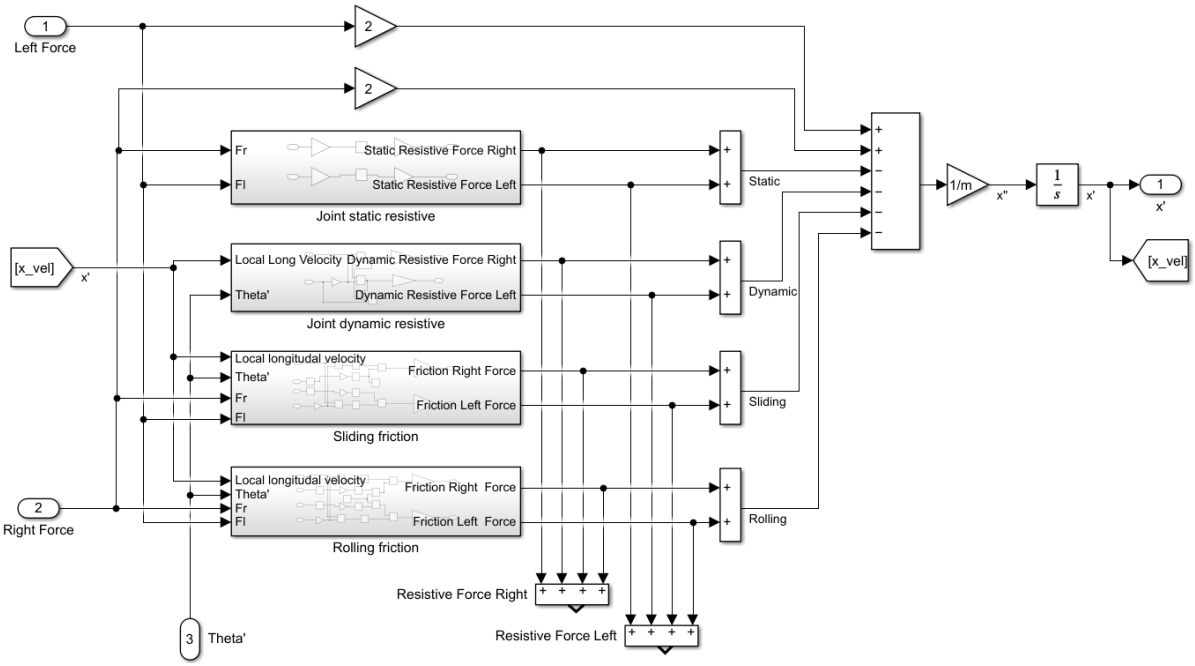


Fig. 3.10: The picture illustrates the Simulink scheme of the longitudinal velocity of the mobile robot in frame R. Inputs are forces on the left (Left Force) and right (Right Force) wheel and the robot's angular velocity (Theta'). Output is longitudinal velocity symbolized as x'.

Change orientation is caused by torque, whose originators are left and right forces reduced by resistive forces acting on half of the robot span t . Resistive torque T_R, which is described in the chapter Dissipative Forces, is caused by lateral sliding friction.

$$I\ddot{\Theta} = (2F_r - F_{RR})t - (2F_l - F_{RL})t - T_R \tag{3.12}$$

The Simulink wired schema is demonstrated in the picture 3.11. By joining two schemas on pictures 3.10 and 3.11 through *Resistive Force Left*, *Resistive Force Right*, and *Theta'* is created a complete model which describes the robot behavior via its longitudinal and angular velocity. The function Matlab Function fcn only determines the size and orientation of the resistive torque. Resistive torque cannot be greater than active torque.

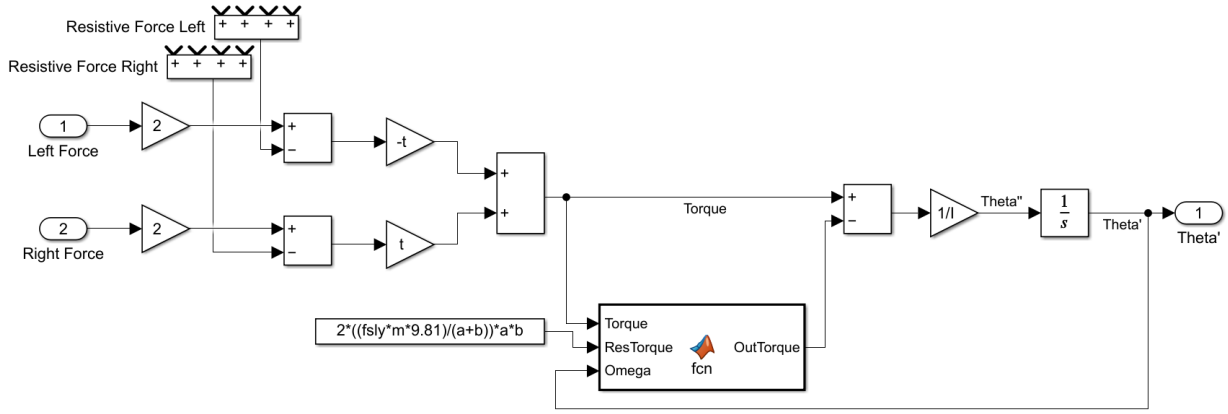


Fig. 3.11: Simulink scheme of angular velocity

According to the equation 3.6, local velocities in frame R can be transformed into global frame G. Subsequent integration of state 3.2 can get the robot's position and orientation.

3.4.1 Dissipative Forces

Equations 3.11 and 3.12 generally describe the longitudinal and angular robot motion. This section will specify all resistive forces used in the Dynamic Model and complete the whole motion model. The model contains four resistive forces with a dominant role in the Gazebo simulator, which can be easily transferred and tuned to a physical mobile robot. The first two forces are related to joint and are set in <gazebo> tags in URDF format. There are static resistive force (friction in <gazebo> tags) and dynamic resistive force (damping in <gazebo> tags). The other two forces are pretty natural, rolling and sliding resistive force.

Friction is a reactive force acting against a force that causes relative motion. The friction force is caused by many different mechanisms, which rely on material, surface proportions, the velocity of the bodies, lubrication, and contact geometry. Before describing the individual forces, some basics properties of friction are mentioned. Static friction depends only on the force's size and is active when velocity is zero.

$$F = \begin{cases} F_e & v = 0 \wedge |F_e < F_s| \\ F_s sgn(F_e) & v = 0 \wedge |F_e| \geq F_s \end{cases} \tag{3.13}$$

The body starts to move after external F_e force overcoming the static friction force F_s.

The next friction model is called Coulomb friction (sometimes called kinetic friction), which only depends on the velocity direction but does not depend on

its size. The velocity v direction is acquired from a math function called signum $sgn(x)$, which has a discontinuity at zero and cannot be differentiated. This problem is solved in the followings paragraphs.

$$F = f_c F_N sgn(v) \tag{3.14}$$

f_c is the Coulomb friction coefficient, which usually depends on materials and is lower than the static friction coefficient f_s.

Viscous friction depends linearly on velocity.

$$F = f_v F_N v \tag{3.15}$$

This type of friction occurs when there is some oil between the contact surface. [57]

Discontinuity function signum caused the impossibility of Jacobian linearization around a near-zero operating point. Therefore, the arctan function approximates the signum function ($k \gg 1$), as shown in the following equation 3.16 and graph 3.12 with different k parameters.

$$\widehat{sgn(x)} = \frac{2}{\pi} \arctan(kx) \tag{3.16}$$

The limit equation 3.17 gives the exact relationship.

$$sgn(x) = \lim_{k \to \infty} \frac{2}{\pi} \arctan(kx) \tag{3.17}$$

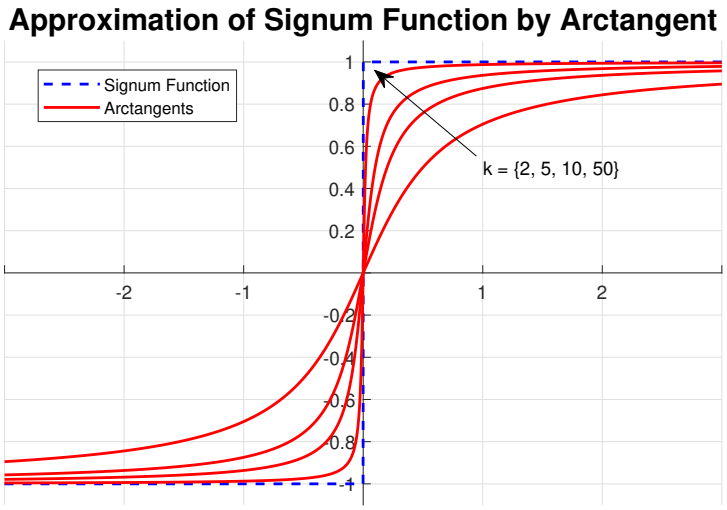


Fig. 3.12: The increasing k (in the direction of the arrow) better approximates the signum function by arctan.

The force F_n from equations one and two is the normal force, which acts on the body and protects the movement through obstacles. The originator of normal forces is the gravitational force. The normal force acts on each wheel can be expressed according to [54] [53] as

$$\sum_{n=1}^{4} F_{Ni} = mg \tag{3.18}$$

Where F_{N1} and F_{N2} are expressed as equations 3.19 and 3.20 for the front and back wheels. The parameters a and b are the distance from COM to the front wheel and back wheel, respectively.

$$F_{N1} = F_{N2} = \frac{b}{a+b} mg \tag{3.19}$$

$$F_{N3} = F_{N4} = \frac{a}{a+b} mg \tag{3.20}$$

Joint Static Friction

The aforementioned joint properties (chapter Physical Properties)in the Gazebo simulator allow setting friction value T_s, which determines the minimum torque $[Nm]$ or force $[N]$ needed to move. The unit depends on the type of joint (prismatic, revolute, continuous, etc.). The behavior of this resistive force is static friction without zero-speed conditions. The static joint friction acts until the input force is stopped. The joint or wheel connected with the joint start to move after input force or torque overcomes the set value. The ideal action of this resistive force is shown in the following graph 3.13 with a dashed line and its approximation with the hyperbolic tangent function.

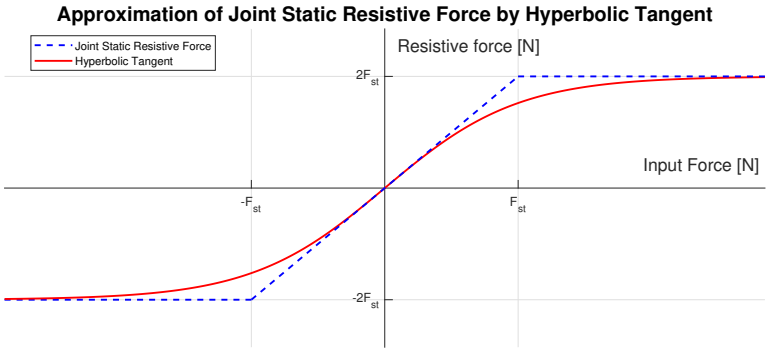


Fig. 3.13: The graph represents the approximation model of Static Joint Friction. On the x-axis is the input force acting on one wheel. Doubled output resistive force is due to applying the same input force on two wheels on each side.

The continuous function approximately represents the static joint friction force of the left F_{sl} and the right F_{sr} side of the robot is shown below.

$$\begin{aligned} F_{sl} &= 2\frac{T_s}{r}\tanh(F_L\frac{r}{T_s}) \\ F_{sr} &= 2\frac{T_s}{r}\tanh(F_R\frac{r}{T_s}) \end{aligned} \tag{3.21}$$

Doubled force $\frac{T_s}{r}$ results from the third assumption - the wheels on each side have the same velocity. Parameter r is wheel radius.

Joint Dynamic Friction

Unlike static joint friction, dynamic joint friction T_d acts on the joint depending on the angular velocity. Its unit is $N\frac{d}{rad}$ or $Nm\frac{s}{rad}$ depending on the type of the joint, same as in previous friction. The angular velocity of wheels can be determined from equations 3.8 and knowledge about wheel radius r. The dynamic resistive forces are

$$\begin{aligned} F_{dl} &= 2\frac{T_d}{r^2}(\dot{x} - \dot{\Theta}) \\ F_{dr} &= 2\frac{T_d}{r^2}(\dot{x} + \dot{\Theta}) \end{aligned} \tag{3.22}$$

The picture shows that the left and right forces are crucial to derive the equation of robot angular motion. But knowing these forces is irrelevant for linear velocity, and the equations above can be joined together.

$$F_d = F_{dr} + F_{dl} = 4\frac{T_{dy}}{r^2}\dot{x} \tag{3.23}$$

Rolling and Sliding

The robot's motion is caused by rolling wheels on the surface. When the robot moves, it can be two types of resistive forces. One of them occurs when the wheel rotates, and no slipping happens. This resistive force caused by deformations and inequality on the surface is called rolling. [5] Rolling friction can be expressed by Coulomb friction or can be neglected because rolling friction is much smaller than other resistance sources. The next not negligible resistive force is sliding friction.

The robot's motion is caused by rolling wheels on the surface. When the robot moves, it can happen two types of resistive forces. The rolling resistive force caused by deformations and inequality occurs when the wheel rotates, and no slipping happens. Rolling friction can be expressed by Coulomb friction or can be neglected

[5] The first assumption from the section 3.3 is omitted

because rolling friction is much smaller than other resistance sources. [6] The next, not negligible resistive force is sliding friction. Sliding friction occurs when the robot moves and the input torque (the force in the model) goes to zero, and the wheel stops. The model of sliding friction can be modeled as viscous friction. The assumption from the 3.3 section does not allow the wheel slippage. This assumption is not considered because the Gazebo experiments demonstrate that wheel slippage occurs when the force is stopped due to the realistic friction coefficient set in the Gazebo model.

Both rolling resistance and sliding friction occur when the robot moves, and the resulting resistive force must be distinguished. Two cases of friction can be determined with the help of input force. The rolling friction is active when the wheels move; therefore, it is active when the input force (torque) overcomes the static joint. In other cases, when the robot has nonzero velocity and input force is less than static friction force. Such a sharp transition between those states is unsuitable for a simplified model. Therefore, a Gaussian function is used to transition from one state to another. It is the use of similarity with part of fuzzy logic.

The following equations derive the properties of the resulting Gaussian functions, which are used in the rolling and sliding friction model. Let's have the simple Gaussian function.

$$f(x) = e^{-kx^2} \tag{3.24}$$

Let the friction force set in the static friction model intersects the Gaussian function at the inflection point. The second derivative of the equation 3.24 is as follows.

$$\frac{d^2f(x)}{dx^2} = 4k^2x^2e^{-kx^2} - 2ke^{-kx^2} \tag{3.25}$$

The inflection point is the solution of the equation.

$$\begin{aligned} \frac{d^2f(x)}{dx^2} &= 0 \\ x_{1,2} &= \pm\frac{\sqrt{2\frac{1}{k}}}{2} \end{aligned} \tag{3.26}$$

The value k depends on F_{st} is equal as follows.

$$k = \frac{1}{2F_{st}^2} \tag{3.27}$$

[6] The first assumption from the section 3.3 is omitted

The rolling resistance can be modeled as

$$F_{RO} = f_{RO}\frac{mg}{2}\left((1-e^{-kF_R^2})sgn(\dot{x}+t\dot{\Theta})+(1-e^{-kF_L^2})sgn(\dot{x}-t\dot{\Theta})\right) \tag{3.28}$$

and sliding friction as

$$F_{SL} = f_{SL}\frac{mg}{2}\left(e^{-kF_R^2}(\dot{x}+t\dot{\Theta})+e^{-kF_L^2}(\dot{x}-t\dot{\Theta})\right) \tag{3.29}$$

When f_{RO} and f_{SL} are rolling friction and sliding friction coefficients, respectively.

$$f_{RO} \ll f_{SL} \tag{3.30}$$

All resistive forces from the equation 3.11 describing the linear velocity of the robot are defined.

Resistive Sliding Torque

The different wheels' velocities on each robot side cause the change orientation. This effect is caused by torque 3.12 induced with different sizes of input force acting on wheels according to the equation. This torque is reduced by resistive forces acting on each wheel described in previous subsections, and sliding movement in a lateral direction acts against torque also with considerable resistive torque T_r.

The Gazebo's robot description does not clearly imply the resistance torque properties. Therefore are two ways how to describe sliding resistive torque, which are compared in the Comparing with Gazebo. The first resistance model does not depend on the angular velocity, only on its signs.

$$T_r = 2f_{SLy}mg\frac{ab}{a+b}sgn(\dot{\Theta}) \tag{3.31}$$

The second model depends on the angular velocity, where

$$T_r = 2f_{SLy}mg\frac{ab}{a+b}\left(\dot{\Theta}+sgn(\dot{\Theta})\right) \tag{3.32}$$

f_{SLy} is the lateral sliding friction coefficient.

3.4.2 Comparing with Gazebo

This section compares the derived model in previous chapters 3.4 with measured data from the Gazebo experiment. It was realized two types of experiments. The first was made to check the correctness of the equation of motion 3.11 in a linear direction. The second was made to check the angular velocity 3.12 correctness,

which, as opposed to experimenting with linear velocities, the robot was driven with the opposite wheel's torque.

Before continuing, the metrics for evaluating the robot model are mentioned. For the two first experiments is used the *RMSE* (Root-mean-square error),

$$RMSE = \frac{1}{N^2}\sqrt{\sum_{k=1}^{N}\left(\hat{y_k} - y_k\right)^2} \tag{3.33}$$

where N denotes the number of measurements and R2 (R squared or the coefficient of determination)

$$R^2 = 1 - \frac{\sum_k \left(y_k - \hat{y_k}\right)^2}{\sum_k \left(y_k - \bar{y_k}\right)^2} \tag{3.34}$$

represents the amount of variation that model explains. $R2$ multiplied by 100 represents the amount in percentage.

In each experiment, the type was realized five experiments. The order of the experiments and applied left and right wheels torque shows in the table 3.1 (it shows the sum torque on both wheels on the side).

Tab. 3.1: Experiments overview

Order		1	2	3	4	5
Linear: Torque [Nm]	L	40	80	120	160	200
	R	40	80	120	160	200
Angular: Torque [Nm]	L	-60	-80	-100	-120	-140
	R	60	80	100	120	140

The results of the first set of experiments show the following graphs 3.14. The only visual evaluation shows a good match model with the simulator. The transient response and steady-state value could correspond to the real process.

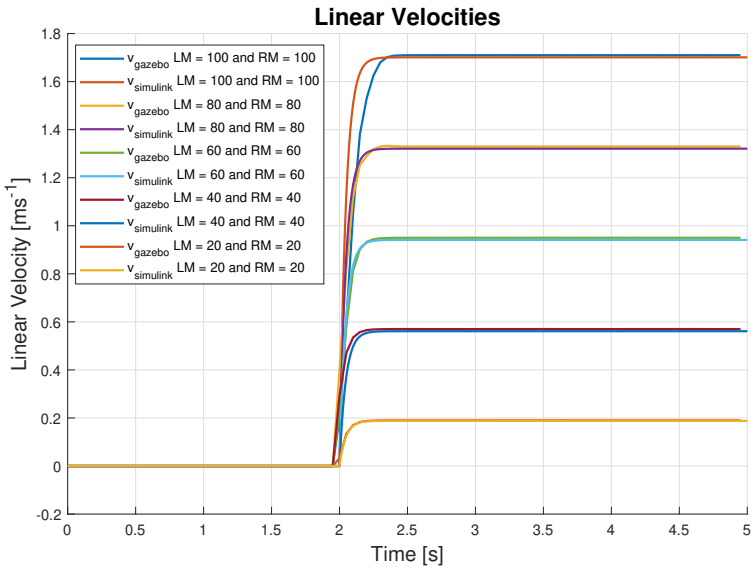
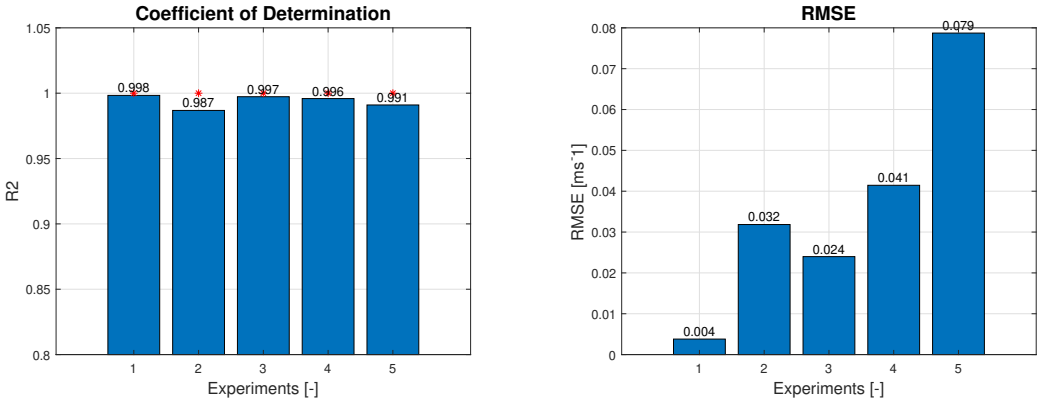


Fig. 3.14: The graph compares the model and robot motion in the Gazebo simulator

The model suitability confirms the R^2 values, which are close to one, while the $RMSE$ values are near to zero 3.15.



(a) (b)

Fig. 3.15: R^2 and $RMSE$ values for linear velocity

The previous section defines two types of resistance models affecting robot motion. The first depends on the resistance model that does not depend on angular velocity size (Coulomb friction model), and the next resistance model depends on the size of angular velocity (combination of Coulomb and viscous friction model). Therefore the second experiment is compared with two models. The experiment results are shown in pictures 3.16 and 3.18. From the first picture is seen that the robot's angular velocity is less correspondent with increasing velocity.

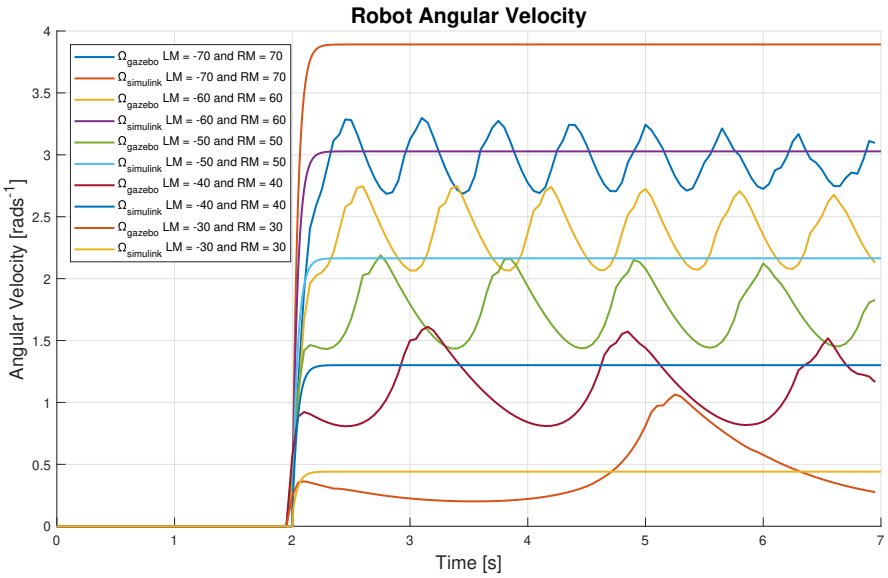
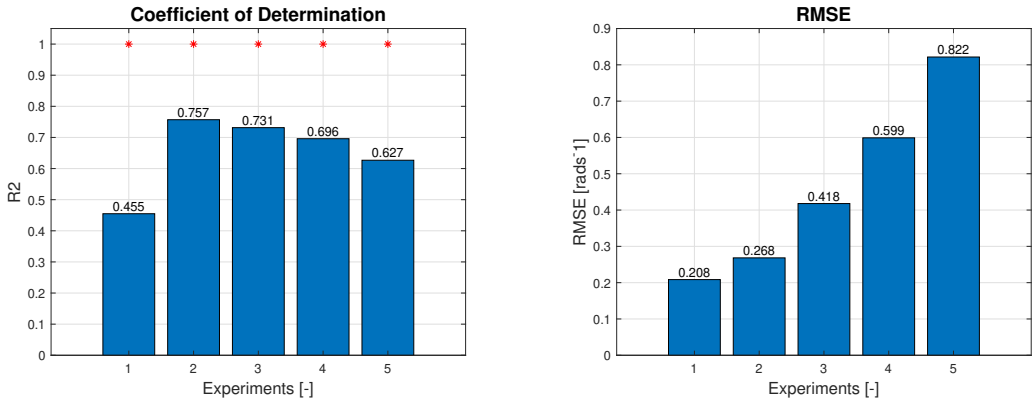


Fig. 3.16: The graph compares the model and angular robot motion in the Gazebo simulator with Coulomb friction resistive model

The angular velocity oscillates during the motion, which is caused by changing the center of rotation. A sharp edge of the wheel could cause this, but the thesis does not deal with it anymore. The R^2 values decrease with increasing velocity. The $RMSE$ values rapidly grow with increasing velocity. The result shows that the resistive sliding torque depends (increases) on the angular velocity.



(a) (b)

Fig. 3.17: $R2$ and $RMSE$ values for angular velocity with Coulomb friction model

Enhanced Coulomb friction to a combination of viscous and Coulomb friction model shows the following results 3.18. With this change, the angular velocity is

modeled correctly. The metrics $RMSE$ and R^2 from the picture 3.19 show significant improvement, except for the R^2 value with the smallest velocity. It is caused by high variance due to inaccurate simulation of robot rotation while the average speed is low, but the $RMSE$ value confirms the correctness of the model resulting from the picture 3.18.

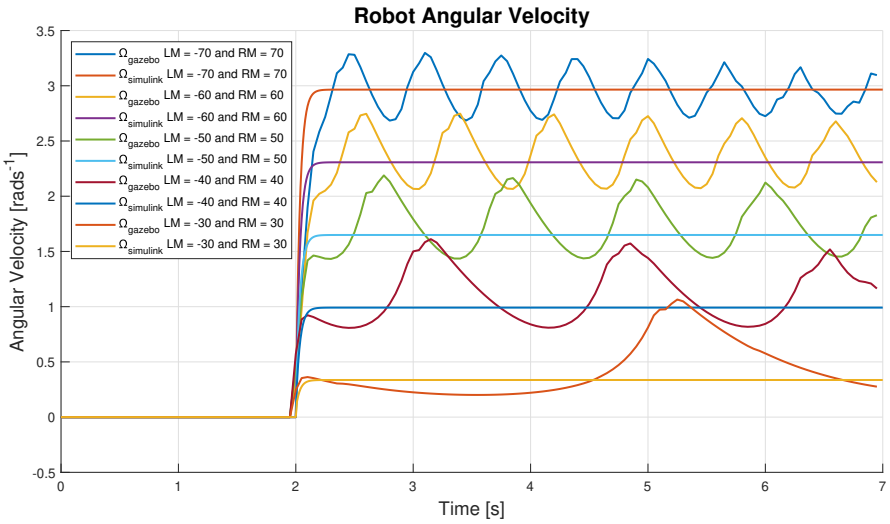
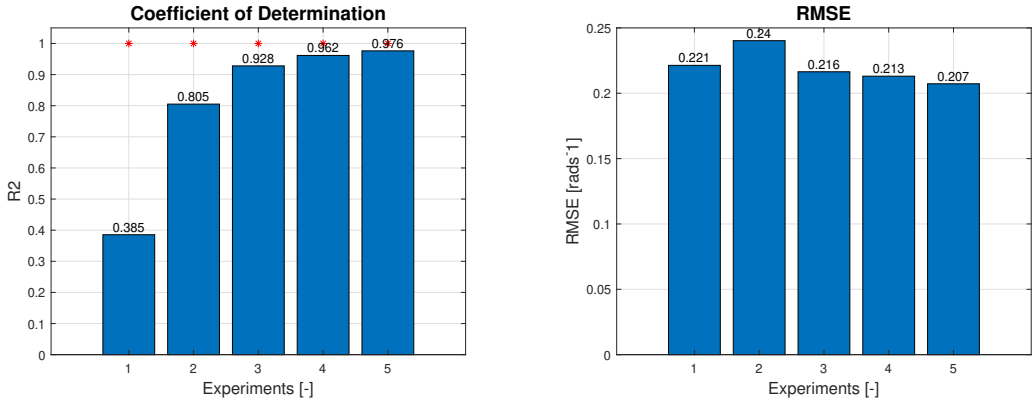


Fig. 3.18: The graph compares the model and angular robot motion in the Gazebo simulator with the enhanced friction model



(a) (b)

Fig. 3.19: $R2$ and $RMSE$ values for angular velocity with a combination of Coulomb and viscous friction model

The model base on the equation is complex and includes too much detail, which does not bring much information relative to the complexity of the model. The follow-

ing section deals with the controls trajectory tracking problem, and the complexity model is solved. Two methods control methods are mentioned.

3.5 Controllers

The movement is a fundamental, essential property of mobile robots. The goal of autonomous robotics is to fulfill some task missions. In the mission process, the trajectory correct movement is a necessary assumption to complete the robotics task without collision. Therefore this research deals with the trajectory tracking problem, whose diagram is shown in the following figure 3.20.

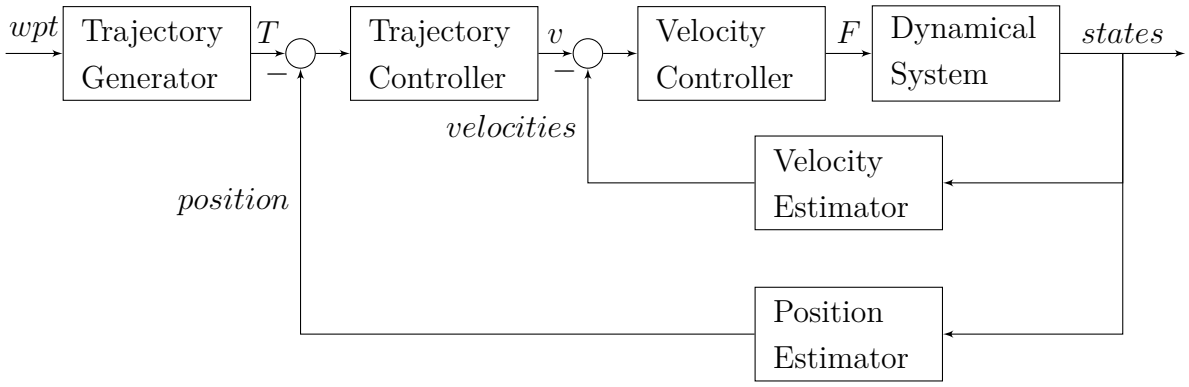


Fig. 3.20: The picture shows the diagram of the trajectory tracking problem solution. Where *wpt* is a set of waypoints, T is generated trajectory, v is desired robot velocity, and action F are forces acting on the robot's wheels.

The diagram 3.20 of the trajectory tracking problem consists of five functionalities or ROS nodes whose cooperation will ensure the movement of the robot on the trajectories. The first application is not shown in the diagram 3.20 because this Controllers does not deal with it. However, after achieving the waypoints, the Trajectory Generator computes the trajectory with respect to kinematics and dynamics restrictions or other metrics that solve different optimal aspects (time, distance, smoothness, and more). The outer feedback loop generates the robot velocities, ensuring the robot's movement to the gained trajectory. The inner loop generates the forces, which guarantees the robot moves with the proper velocity. The picture shows 3.20 the Dynamical System described in the previous section, and Estimators estimate the dynamical system states. Estimators are not considered in this thesis.

The following sections Trajectory Generator, Trajectory Controller, and Velocity Controller describe the individual functionality mentioned above.

3.5.1 Trajectory Generator

The picture 3.21 augments the trajectory tracking diagram 3.20 in the previous chapter. The waypoints can be obtained from a higher level using the robot or as a control input.

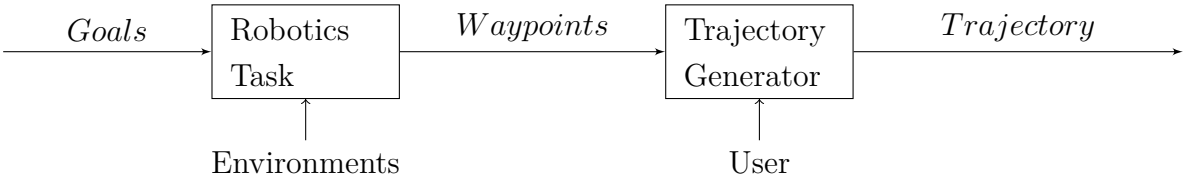


Fig. 3.21: The diagram of the trajectory generator in cooperation with inputs and higher levels of planning

This thesis generates trajectory from waypoints as the generally known *cubic spline*. Let us have a set of waypoints W in the XY plane and

$$W = \{[X_0, Y_0], [X_1, Y_1], \ldots, [X_n, Y_n]\} \tag{3.35}$$

time set T.

$$T = \{t_0, t_1, \ldots, t_n\} \tag{3.36}$$

The cubic spline is derived only for X coordinates W_x without loss of generality.

$$W_x = \{X_0, X_1, \ldots, X_n\} \tag{3.37}$$

The cubic spline is based on finding n polynomial functions piecewise between W_x points. The name cubic suggests the polynomial has third order.

$$\begin{aligned} f_1(t) &= a_1t^3 + b_1t^2 + c_1t + d_1 \qquad t \in \langle t_0, t_1 \rangle \\ f_2(t) &= a_2t^3 + b_2t^2 + c_2t + d_2 \qquad t \in \langle t_1, t_2 \rangle \\ &\vdots \\ f_n(t) &= a_nt^3 + b_nt^2 + c_nt + d_n \qquad t \in \langle t_{n-1}, t_n \rangle \end{aligned} \tag{3.38}$$

The first requirement for these functions is that their border points in the given times are equal with W_x.

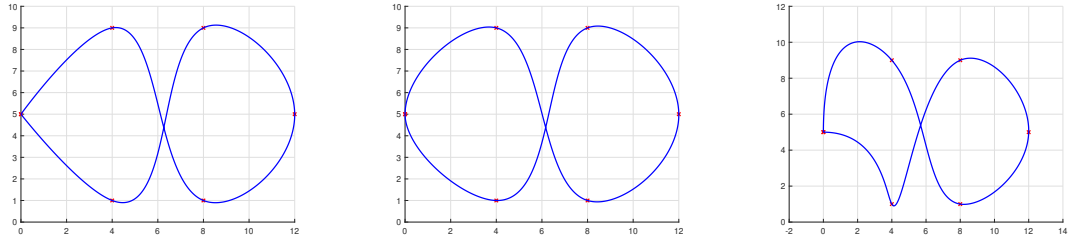
$$
\begin{aligned}
f_1(t_0) &= X_0 \\
f_1(t_1) &= X_1 \\
f_2(t_1) &= X_1 \\
f_2(t_2) &= X_2 \\
&\vdots \\
f_n(t) &= X_n
\end{aligned}
\tag{3.39}
$$

Sustain the continuity of motion is defined as a requirement for first and second derivatives. The derivatives of two functions 3.38 for points $X_1, \ldots, X_{n-1}$ must be equal.

$$
\begin{aligned}
\frac{d}{dt} f_1(t_1) &= \frac{d}{dt} f_2(t_1) \\
\frac{d^2}{dt^2} f_1(t_1) &= \frac{d^2}{dt^2} f_2(t_1) \\
\frac{d}{dt} f_2(t_2) &= \frac{d}{dt} f_3(t_2) \\
\frac{d^2}{dt^2} f_2(t_2) &= \frac{d^2}{dt^2} f_3(t_2) \\
&\vdots \\
\frac{d}{dt} f_{n-1}(t_{n-1}) &= \frac{d}{dt} f_n(t_{n-1}) \\
\frac{d^2}{dt^2} f_{n-1}(t_{n-1}) &= \frac{d^2}{dt^2} f_n(t_{n-1})
\end{aligned}
\tag{3.40}
$$

Four coefficients of each spline have a $4n$ equations that must be defined for a successful solution. Which $2n$ equations define equality at points, and $n-1$ defines first and second derivatives, respectively. The last two-equation solves the initial and end conditions for velocities.

$$
\begin{aligned}
\frac{d}{dt} f_1(t_0) &= v_{start} \\
\frac{d}{dt} f_n(t_n) &= v_{end}
\end{aligned}
\tag{3.41}
$$



(a) Zero conditions

(b) Nonzero symmetric conditions

(c) Nonzero asymmetric conditions

Fig. 3.22: Example of cubic spline trajectory with different initial conditions

3.5.2 Trajectory Controller

The trajectory controller computes reference velocities satisfying the input trajectory. This thesis implements two types of trajectory controllers. The first controller [59] [60] is described below, and the second controller is based on a Linear-quadratic regulator 3.5.3.

The following equation shows the mutual relationship between desired robot velocity in the coordinate system G 3.3 and linear and angular speed,

$$\begin{pmatrix} \dot{X}_d + i_x \\ \dot{Y}_d + i_y \end{pmatrix} = \begin{pmatrix} \cos\Theta & D\sin\Theta \\ \sin\Theta & -D\cos\Theta \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{3.42}$$

where D is the difference between COG and COM and i_x, i_y is the control action to eliminate disturbances. After matrix inversion, the control input is obtained from 3.42.

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \cos\Theta & \sin\Theta \\ -\frac{1}{D}\sin\Theta & \frac{1}{D}\cos\Theta \end{pmatrix} \begin{pmatrix} \dot{X}_d + i_x \\ \dot{Y}_d + i_y \end{pmatrix} \tag{3.43}$$

$$i_x = l_x \tanh \frac{k_x}{l_x}\tilde{X} \tag{3.44}$$

$$i_y = l_y \tanh \frac{k_y}{l_y}\tilde{Y} \tag{3.45}$$

The $\tilde{X}$ and $\tilde{Y}$ are defined as the distance between the actual position and desired trajectory.

$$\begin{aligned} \tilde{X} &= X - X_d \\ \tilde{Y} &= Y - Y_d \end{aligned} \tag{3.46}$$

l_x and l_y define maximum tracker action to remove position and desired trajectory inaccuracies. k_x and k_y determine the slope of the $\tanh x$ function.

3.5.3 Velocity Controller

According to chapters Motion Analysis and Dynamic Model, the robot model shown in the pictures 3.23 3.25 3.27 3.29 3.31 can be described by following the nonlinear equation, representing the states X, Y, Θ in reference G and its local linear and angular velocity Ω and v.

$$\begin{aligned}
\dot{X} &= v\cos\Theta = f_1(\vec{x},\vec{u}) \\
\dot{Y} &= v\sin\Theta = f_2(\vec{x},\vec{u}) \\
\dot{\Theta} &= \Omega = f_3(\vec{x},\vec{u}) \\
\dot{\Omega} &= \frac{2F_r - F_{RR}}{I}t - \frac{2F_l - F_{RL}}{I}t - \frac{T_R}{I} = f_4(\vec{x},\vec{u}) \\
\dot{v} &= \frac{2F_l + 2F_r}{m} - \frac{F_{static} - F_{dynamic} - F_{sliding} - F_{rolling}}{m} = f_5(\vec{x},\vec{u})
\end{aligned} \tag{3.47}$$

This nonlinear dynamic system can be divided into a simplified linear dynamic and a kinematic system. Linearized dynamic systems contain the robot's linear and angular velocities. Its equation in state-space description format

$$\begin{aligned}
\dot{\vec{x}} &= \mathbf{A}\vec{x} + \mathbf{B}\vec{u} \\
\vec{y} &= \mathbf{C}\vec{x} + \mathbf{D}\vec{u}
\end{aligned} \tag{3.48}$$

is as follows.

$$\begin{aligned}
\begin{pmatrix}\dot{\Omega}\\ \dot{v}\end{pmatrix} &= \begin{pmatrix}-4\frac{T_d}{r^2 I}t^2 - 2\frac{f_{SLy}mg}{(a+b)I}ab & 0\\ 0 & -4\frac{T_{dy}}{r^2 m}\end{pmatrix}\begin{pmatrix}\Omega\\ v\end{pmatrix} + \begin{pmatrix}-\frac{2t}{I} & \frac{2t}{I}\\ \frac{2}{m} & \frac{2}{m}\end{pmatrix}\begin{pmatrix}F_l\\ F_r\end{pmatrix} \\
\vec{y} &= \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}\omega\\ v\end{pmatrix}
\end{aligned} \tag{3.49}$$

And kinematic part of the system is

$$\begin{aligned}
\begin{pmatrix}\dot{X}\\ \dot{Y}\\ \dot{\Theta}\end{pmatrix} &= \begin{pmatrix}\cos\Theta & 0\\ \sin\Theta & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}v\\ \Omega\end{pmatrix} \\
\mathbf{y} &= \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}X\\ Y\\ \Theta\end{pmatrix}
\end{aligned} \tag{3.50}$$

The equation 3.50 is valid when input forces are much greater than joint static resistive. In these assumptions, the joint static forces can be neglected, like rolling friction. The sliding friction is zero when the forces are active; Therefore can also be neglected. The simplification in the equation for angular velocity counts with

the joint static resistive, rolling, and sliding forces do not cause torque and therefore are not included in F_{RR} and F_{RL}.

For such a simplified system can be designed the linear controllers. This section deals with designing controllers, and the trajectory tracking problem is finished.

Discrete PI

PID (**P**roportional **I**ntegral **D**erivative) feedback control law is highly used in industry for there simplicity, intuitive tuning, and robustness. Not all parts of PID must be contained; therefore, other derivatives such as P, PI, or PD controller are used. The first controller discussed in the thesis is the PI controller. The following equations describe the PI controller in discrete time with step k and Z-transform.

$$u_{k+1} = K\left(e_k + \frac{T_s}{T_i}\sum_{i=1}^{k} e_k\right) \tag{3.51}$$

$$G(z) = K\left(1 + \frac{T_s}{T_i}\frac{z^{-1}}{1 - z^{-1}1}\right) \tag{3.52}$$

The control action output is based on tracking error, the difference between desired and actual values. The following structure 3.23 of Simulink control feedback implementation shows that two PI controller commands wheels on each side (the wheel's velocity is the same on each side).

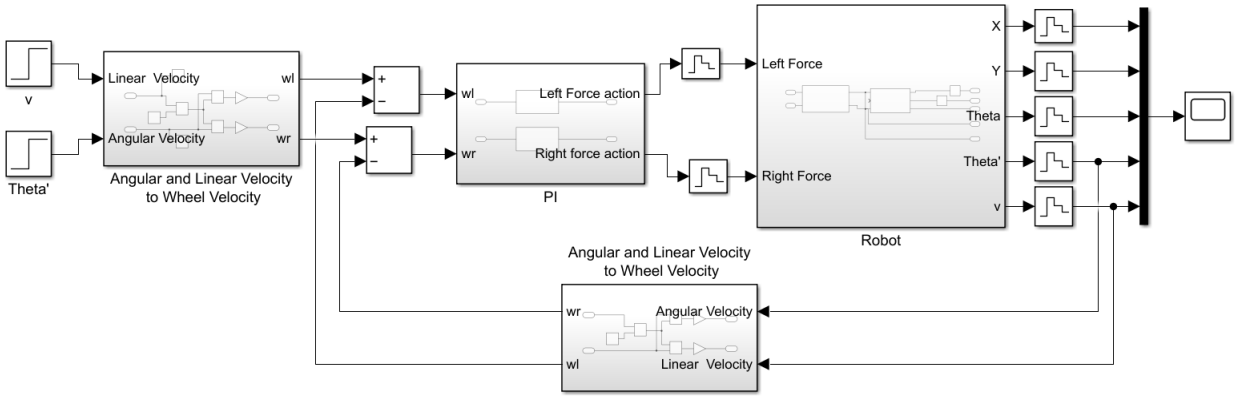
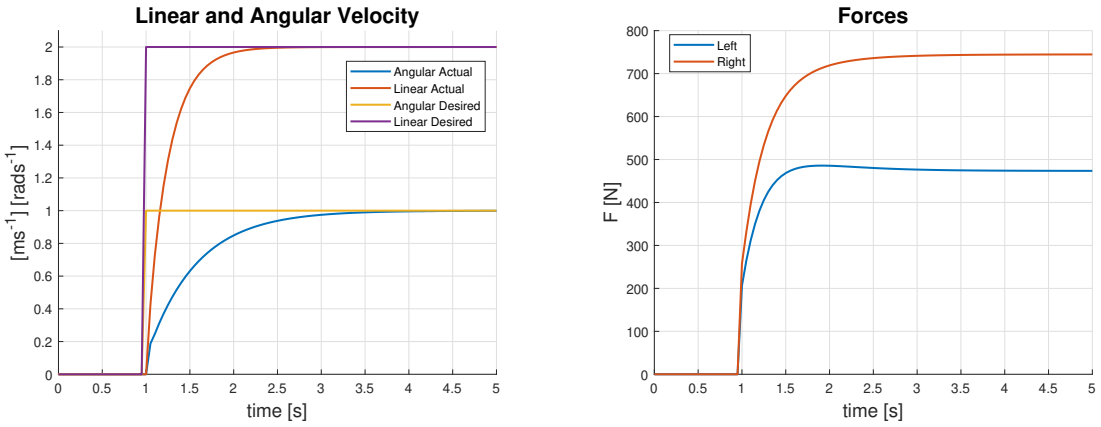


Fig. 3.23: Feedback loop with PI controller

The output of the velocity controller is desired linear and angular velocity. The kinematic equation 3.8 calculates the desired wheel's velocity, the same as the robot's actual wheel's velocity.

The following step response presents linear and angular velocity steps and applied wheels forces. The PI controller is set up to reduce overshot, which implies a slower

angular velocity response. A slower angular velocity response brings a problem with a sharply curved trajectory.



(a) (b)

Fig. 3.24: The graphs show linear and angular velocity responses and applied force to the wheels - PI

Discrete LQR

Linear quadratic regulator (LQR) [61] [62] [63] is a state feedback controller based on solving optimal quadratic criterion 3.61. The controller ensures the best state behavior with respect to a predefined criterion, penalizing input and state. As the name suggests, the LQR is suitable for linear or linearized dynamic systems.

This section derives a linear quadratic regulator with each step linearization of the dynamic system at a non-equilibrium point.

The linearized nonlinear dynamic system $\vec{f}(\vec{x},\vec{u})$ can be described in the following form around the actual state $\vec{x_0}$ and actual input $\vec{u_0}$.

$$\vec{f}(\vec{x},\vec{u},t) = \vec{f}(\vec{x_0},\vec{u_0},t) + \frac{\partial \vec{f}}{\partial \vec{x}}\Big|_0 (\vec{x}-\vec{x_0}) + \frac{\partial \vec{f}}{\partial \vec{u}}\Big|_0 (\vec{u}-\vec{u_0}) \tag{3.53}$$

$$\begin{aligned} \vec{f}(\vec{x_0},\vec{u_0},t) &= \dot{\vec{x_0}} \\ \vec{f}(\vec{x},\vec{u},t) &= \dot{\vec{x}} \end{aligned} \tag{3.54}$$

The system can be rewritten into a state-space description with the following denotation.

$$\frac{\partial \vec{f}}{\partial \vec{x}}\Big|_0 = \mathbf{A} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\vec{x}=\vec{x_0}} \qquad \frac{\partial \vec{f}}{\partial \vec{u}}\Big|_0 = \mathbf{B} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \vdots & \ddots & \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\vec{u}=\vec{u_0}} \tag{3.55}$$

$$\dot{\vec{x}} - \dot{\vec{x_0}} = \mathbf{A}(\vec{x} - \vec{x_0}) + \mathbf{B}(\vec{u} - \vec{u_0}) \tag{3.56}$$

The new system state $\delta\vec{x}$ and input $\delta\vec{u}$ are the difference between the future and the actual state and input, respectively.

$$\begin{aligned} \delta\vec{x} &= \vec{x} - \vec{x_0} \\ \delta\vec{u} &= \vec{u} - \vec{u_0} \end{aligned} \tag{3.57}$$

The equation 3.58 shows the linearized dynamic system describing the behavior in the robot's state vicinity.

$$\dot{\delta\vec{x}} = \mathbf{A}\delta\vec{x} + \mathbf{B}\delta\vec{u} \tag{3.58}$$

The continuous-time state-space model is discretized with the following approximation named Euler method.

$$e^{\mathbf{A}t} \approx \mathbf{I} + \mathbf{A}dt \tag{3.59}$$

The linear quadratic regulator is based on the resulting linearized discrete state-space model 3.60.

$$\delta\vec{x_{k+1}} = (\mathbf{I} + \mathbf{A}dt)\delta\vec{x_k} + dt\mathbf{B}\delta\vec{u_k} \tag{3.60}$$

The behavior of feedback systems 3.25 is affected by different settings $\mathbf{Q}$ and $\mathbf{R}$. The matrix $\mathbf{Q}$ penalizes the deviations of states, and the $\mathbf{R}$ matrix penalizes the control input energy. The criterion of LQR shows the following equation.

$$J = \sum_{k=0}^{N} (\vec{x}^\intercal \mathbf{Q}\vec{x} + \vec{u}^\intercal \mathbf{R}\vec{u}) \tag{3.61}$$

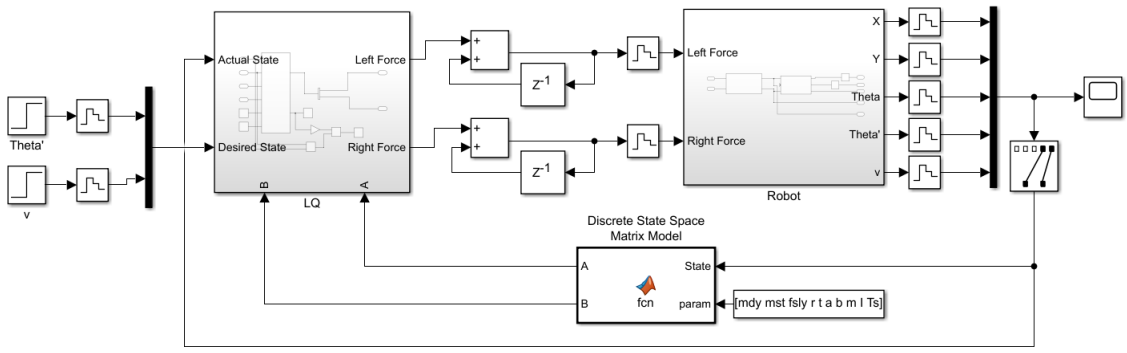


Fig. 3.25: Feedback loop with LQR

Control input is computed according to the equation, where $\delta u_k = u_k - u_k^0$ is the change of input control and $\vec{x_k^d}$ is desired state.

$$\delta\vec{u_k} = -\mathbf{K}\delta\vec{x_k} = -\mathbf{K}(\vec{x_k^d} - \vec{x_k^0}) \tag{3.62}$$

Time invariant $\mathbf{K}$ is defined as

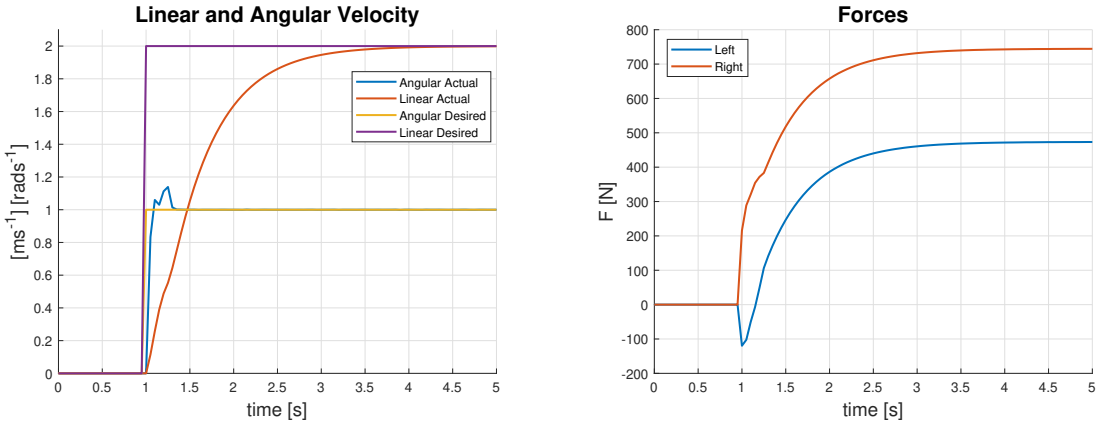
$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^\intercal\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^\intercal\mathbf{P}\mathbf{A} \tag{3.63}$$

and $\mathbf{P}$ solves the discrete algebraic Riccati equation (DARE) [64].

$$\mathbf{P} = \mathbf{A}^\intercal\mathbf{P}\mathbf{A} - (\mathbf{A}^\intercal\mathbf{P}\mathbf{B})(\mathbf{R} + \mathbf{B}^\intercal\mathbf{P}\mathbf{B})^{-1}(\mathbf{B}^\intercal\mathbf{P}\mathbf{A}) + \mathbf{Q}; \tag{3.64}$$

The computing matrix $\mathbf{K}$ is an iterative process.

The velocity step response and the process of the applied force are shown in the following pictures 3.26. The forces are much different from the PI controller. The linear velocity has a slower transient than the PI controller, but the angular velocity reaches the desired value faster than PI. The angular velocity step response has an overshoot, but it can be permitted with respect to fast transient.



(a) (b)

Fig. 3.26: The graphs show linear and angular velocity responses and applied force to the wheels - LQR

3.5.4 Trajectory Tracking

The trajectory tracking problem is supposed to ensure the movement robot on the trajectory. It combines the Velocity Controller as the inner loop and the Trajectory Controller as the outer loop. This section presents three combinations of controllers, T-PI, LQ-PI, and LQ-LQ.

T-PI

T-PI is a trajectory tracking controller which combines the trajectory controller described in the section 3.5.2 and PI 3.5.3 for velocity control. The diagrams 3.27 3.29 3.31 of all three combinations is similar and differs only in controller types. In the Simulink diagram 3.27, the Cubic Spline Trajectory block computes desired trajectory, which is input to the trajectory controller.

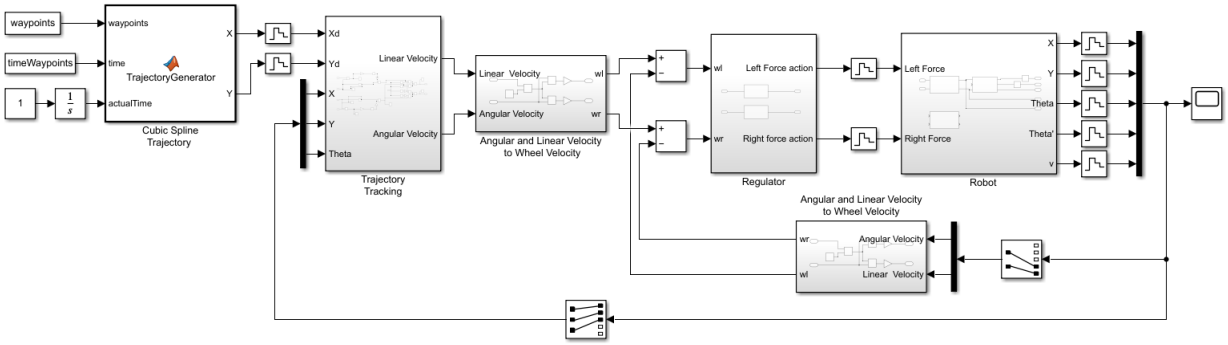
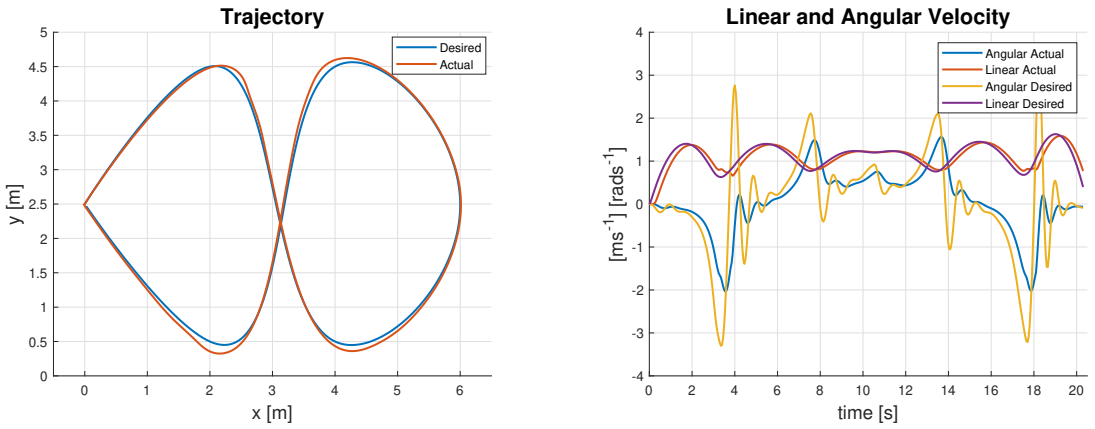


Fig. 3.27: Simulink diagram of T-PI trajectory tracking controller

Trajectory controller based on hyperbolic tangent 3.5.2 has a similar property as P controller 3.5.3 in small desired and actual differences. The hyperbolic tangent controller with proportional action and the slow response of the PI controller 3.5.3 to change angular velocity causes the oscillation of angular velocity 3.28b.



(a) (b)

Fig. 3.28: T-PI: Desired and actual (true) trajectory and linear and angular velocity

By joining a PI controller with a slow angular velocity transient, the robot overshoots the desired trajectory when the angular velocity is changed highly. The overshoot is visible in the graph 3.28a.

LQ-PI

LQ-PI trajectory tracking controller differs from the previous controller 3.5.4 in the trajectory controller part. The hyperbolic tangent controller is replaced with LQR 3.5.3, which is based on the kinematic part 3.50 of the system described in the section Velocity Controller.

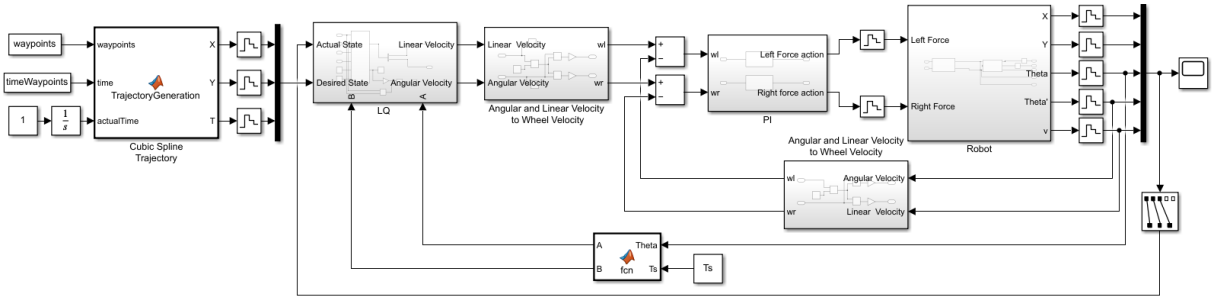
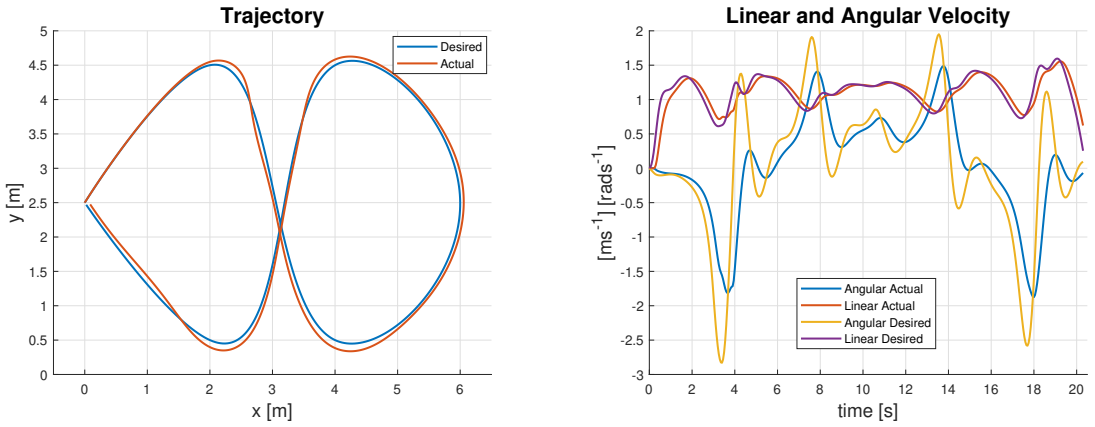


Fig. 3.29: Simulink diagram of LQ-PI trajectory tracking controller

The oscillation of angular velocity is stopped 3.30b. The controller is less aggressive, but the overshoots have remained. The oscillation of angular velocity is stopped 3.30b. The controller is less aggressive, which causes the slower transition to steady-state value 3.30a, but the overshoots have remained.



(a) (b)

Fig. 3.30: LQ-PI: Desired and actual (true) trajectory and linear and angular velocity

LQ-LQ

The last trajectory tracking controller consists of two LQRs. The diagram is shown in the following picture 3.31.

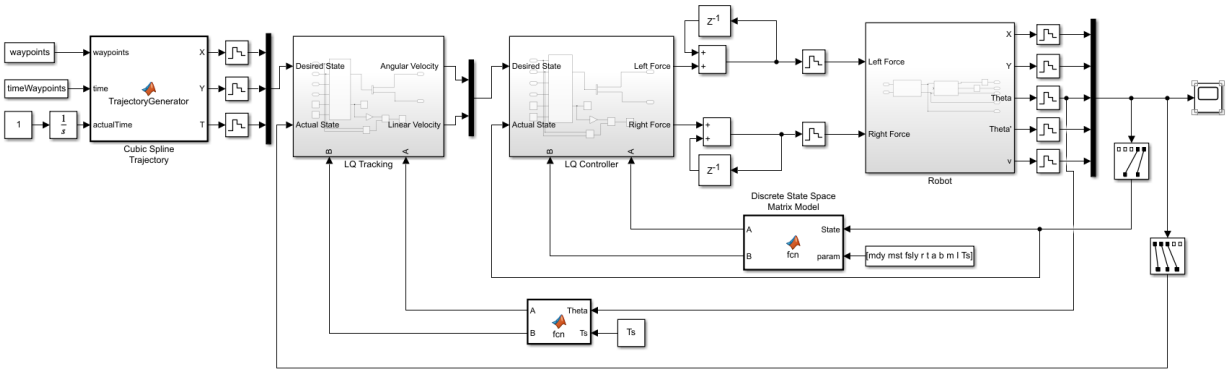
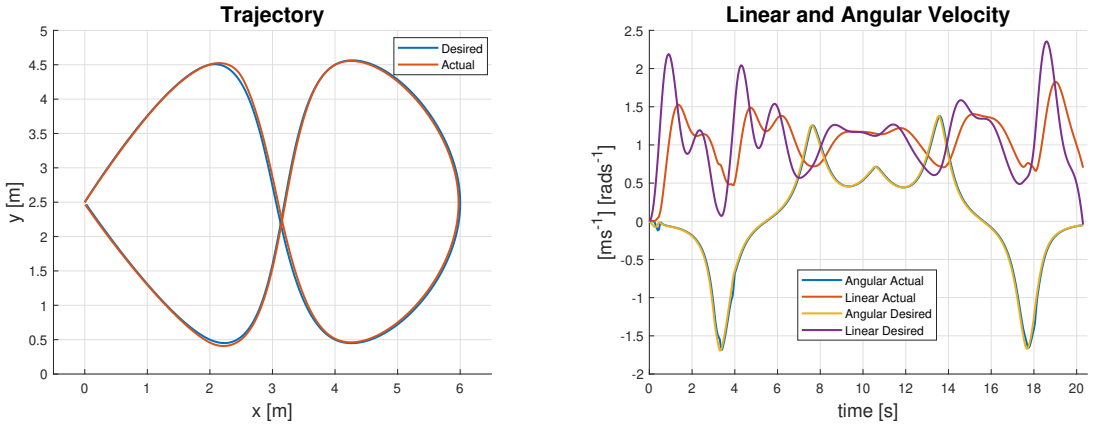


Fig. 3.31: Simulink diagram of LQ-LQ trajectory tracking controller

The velocity LQ controller has a fast angular velocity response 3.26 that solves the problem with trajectory overshoots. The picture 3.32a shows precise trajectory tracking, but linear velocity response is slower 3.32b than the previous two controllers. Therefore the shape of desired and actual trajectory is significantly similar, but the timed difference between the actual and desired position is greater than the previous two controllers.



(a) (b)

Fig. 3.32: LQ-LQ: Desired and actual (true) trajectory and linear and angular velocity

3.6 Coordinate System

A coordinate system is a set of reference frames that determines the position and orientation of dynamic systems, mobile or flying robots, usually. The references frame can be defined in many ways; therefore, the need is formed to define the standard called REP 105 3.6.1.

3.6.1 REP 105

REP105 is a coordinate system convention used in ROS applications. It uses three main types of frames summarized in the following list.

- base-link
- odom
- map

Base-link is a frame joined with a robot body. It can be placed everywhere in the robot, but it is recommended to fulfill other conventions REP103[7].

Odom is a world-fixed frame with the beginning on the spawn/started place of the robot. The robot, base-link, moves in the odom frame without any bounds but must change continuously without discrete jumps. Therefore is used for odometry based on wheels encoder or inertial measurement unit.

The map is a world-fixed frame placed on the coordinate frame origin. The Z-axis is pointing upward. This frame cannot be continuously changed, unlike the odom frame, and can be used GPS measurement with discrete jumps measurement, or use updating algorithm to estimate global long term position.

The following image 3.33 shows the mutual relationship above mentioned frames.

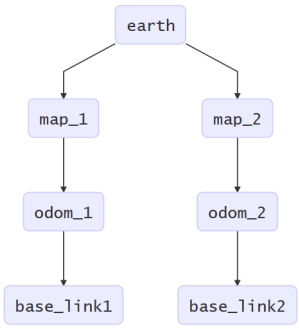


Fig. 3.33: Multi-robot relationship between frames in ECEF [20]

[7] http://www.ros.org/reps/rep-0103.html

The first frame, earth, is used when multiple robots are used in different map frames. It defines the position of the robot in the ECEF (**E**arth-**c**entered, **E**arth-**f**ixed) coordinate frame in the vicinity of the Earth. The following 3.6.2 section describes transferring geodetic coordinates to the local tangent plane ENU coordinate.

3.6.2 Conversion of Geographic Coordinate System

The map is an abstract local tangent plane coordinate, which can be set at a given/reference point in ECEF, as the picture 3.33 from the previous chapter 3.6.1 shows. This section describes the transformation between geodetic coordinates from GPS to position on the map.

The problem shows the picture 3.34.

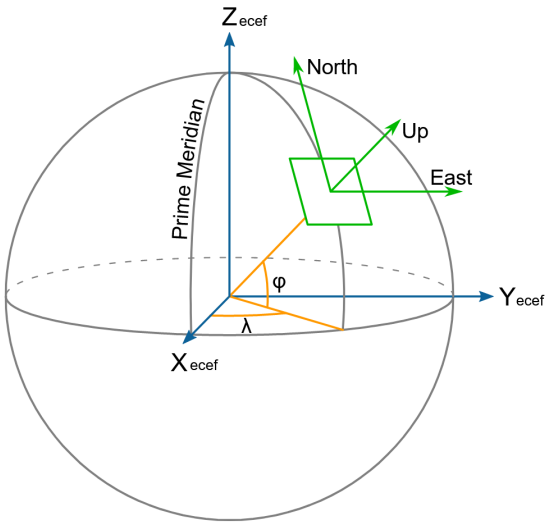


Fig. 3.34: ECEF - Earth-centered, Earth-fixed coordinate system [20]

The following equation determines the transformation from geodetic coordinates to ECEF, where ϕ is latitude, λ is longitude, and h is height. [58]

$$\begin{aligned} X &= \left(\frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} + h \right) \cos\phi \cos\lambda \\ Y &= \left(\frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} + h \right) \cos\phi \sin\lambda \\ Z &= \left(\frac{b^2}{a^2} \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} + h \right) \sin\phi \end{aligned} \tag{3.65}$$

The constant $a = 6378100m$, $b = 6356800m$ is an equatorial radius (semi-major axis) and polar radius (semi-minor axis), respectively. The constant e^2, which results from the aforementioned radius, is the square of the eccentricity of the ellipsoid.

$$e^2 = 1 - \frac{b^2}{a^2} \tag{3.66}$$

The map frame has its origin in a reference point X_m, Y_m, Z_m in the ECEF coordinates. The body link is denoted as X, Y and Z. The local coordinates x, y and z in the map frame are computed according to the equation 3.67.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\sin\lambda_m & cos\lambda_m & 0 \\ -\sin\phi_m\cos\lambda_m & -\sin\phi_m\sin\lambda_m & \cos\phi_m \\ \cos\phi_m\cos\lambda_m & \cos\phi_m\sin\lambda_m & \sin\phi_m \end{pmatrix} \begin{pmatrix} X - X_m \\ Y - Y_m \\ Z - Z_m \end{pmatrix} \tag{3.67}$$

4 Radioactivity Simulation

The testing algorithm on real hardware in the real world is challenging and expensive, as mentioned in the Robotic Simulators chapter. In addition, the experiments with radiation sources must fulfill all legislative conditions, which extends and complicates the experiment's preparation. Therefore, the simulator is again suitable for this task. This thesis used the gazebo_radiation_plugin[1] described in *Simulating Ionising Radiation in Gazebo for Robotic Nuclear Inspection Challenges* [5]. This package is coded for ROS1. Therefore the package was recoded for use with ROS2.

4.1 Radiation Gazebo Plugin

According to [5], the radiation source is described in Cartesian space with an activity value in Becquerels (Bq). This assumption applies if all radiation decay transmits only the gamma radiation from the source. Without these assumptions, the source can be described with sensor counts per second (CPS) in one meter from the source. The physical model is based on the inverse square relationship 1.26 regarding the sensor dimensions (fixed value r = $1 \times 10^{-2} m$).

$$CPS_d = \frac{CPS_0}{2}(1 - \frac{d}{\sqrt{r^2 + d^2}}) \tag{4.1}$$

CPS_d is the estimated value of the radiation sensor in the distance d from the source. If the distance $d \to 0$ the $CPS_{d \to 0}$ goes to the initial activity value CPS_0 divided by two. It is a better reflection of physical reality than inverse square law 1.26.

The above equation 4.1 is valid if the source was defined as the activity value CPS_0 with the aforementioned assumptions (CPS_0 is the theoretical sensor's value that captures all of the gamma rays from the source). If the CPS_1 value is known, the equation is corrected 4.2. CPS_1 is estimated from the equation 4.1.

$$CPS_d = \frac{CPS_1}{1 - \frac{1}{\sqrt{r^2+1^2}}}(1 - \frac{d}{\sqrt{r^2 + d^2}}) \tag{4.2}$$

The intensity of radiation is strongly correlated by distance from the source. But the intensity is also affected by attenuation when gamma rays pass through the medium. The equation describes attenuation behavior. In gazebo_radiation_plugin is linear attenuation coefficient α are entered with unit m^{-1}. The sensor may not scan values uniformly or can be collimated. The arbitrary sensitivity function $\eta(\theta)$ with the angle θ parameter between the source and the sensor returns the value

[1] `https://github.com/EEEManchester/gazebo_radiation_plugin`

between zero and one. This sensitivity function guarantees a directional characteristic. The final value 4.3 on the sensor is given by the sum of contributions from all sources [5].

$$CPS = \sum_{i}^{\infty}(\eta_i \frac{CPS_1}{1 - \frac{1}{\sqrt{r^2+1^2}}}(1 - \frac{d_i}{\sqrt{r^2 + {d_i}^2}}) \prod_{j}^{\infty} e^{-\alpha_j z_j}) \tag{4.3}$$

The z_j value is the distance which gamma radiation travels through the j material.

One of the largest disadvantages of modeling radiation sources and sensors by an approach described in this is the lack of energy spectrum of the radiation source and its effect on the linear attenuation coefficient 1.27. The next large disadvantage is the absence of death time and nonlinear characteristics of the radiation sensor. The two following experiments show the functionality of the sensor in two cases. The sensor does not have a dead time in the first case, and the robot doesn't have an attenuation coefficient. The second experiment is simulated with a dead time of sensor and robot have nonzero attenuation experiment. The goal of the second experiment is approximate the behavior of a real scintillation detector 2x2"NaI(Tl).

4.1.1 Gazebo Experiment

The first experiment was performed to verify the features of the Radiation Gazebo Plugin. The experiment consists of one radiation source, Co-60, whose radiation decay scheme is shown in the picture below 4.1.

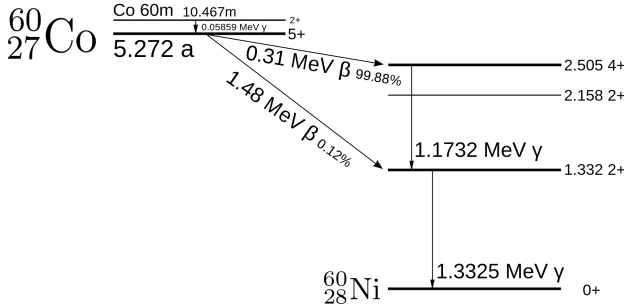


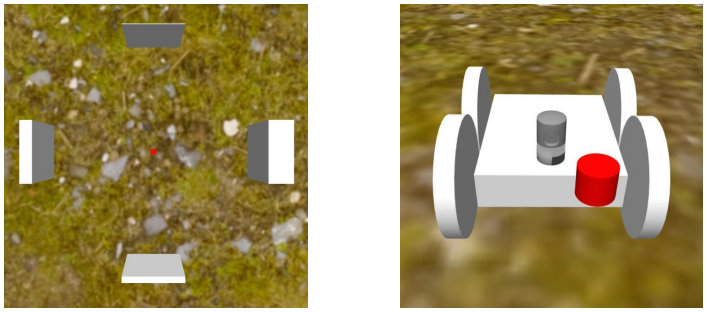
Fig. 4.1: The decay scheme of Cobalt-60 [49]

The energy of gamma rays is important to choose the right linear attenuation coefficient. The most emitted gamma radiation is chosen from the radiation decay scheme with energy $1.1732MeV$, and the simulated source has 10000 counts per second in a $1m$ distance value. The obstacles are from ordinary concrete[2] with

[2] `https://physics.nist.gov/PhysRefData/XrayMassCoef/ComTab/concrete.html`

density $2.4gm^{-3}$, and its linear attenuation coefficient was obtained and recalculated from mass attenuation coefficient in web page[50]. The barrier and the experiment layout is shown in the picture 4.2a. The barrier has in order right, left, down, up the value 40cm, 20cm, 10cm, 1cm. All of them are from the concrete with the same linear attenuation coefficient ($\mu = 13.937m^{-1}$). Inside the square from barriers is a radiation source represented by a red cube.

The sensor is attached to teleoperated Orpheus X4 robot model4.2b. The sensor does not have some directional characteristics. Therefore the sensitive function is disabled. Trajectory covers the entire space with a distance $0.5m$ between parallel roads.



(a) (b)

Fig. 4.2: The layout of the experiment. On the left picture are four obstacles from concrete (1cm, 10cm, 20cm, 40cm) and in the center is a red cube radiation sensor. The right picture shows Orpheus X4 with Velodyne Lidar and a radiation sensor.

Data from the radiation sensor was recorded with command-line tool ros2 bag[3] and interpolated with scatteredInterpolant[4] Matlab [51] function. The picture 4.3 show interpolated data from the radiation sensor with a resolution $0.01m$, and the axis orientation is the same as in the picture above. The sensor is in height 0.19. Therefore the circle with $10000cps$ is closer than a circle about a one-meter radius from the start. The picture 4.3 shows an ideal sensor's theoretical values capture all passing particles or the true radiation intensity.

[3] `https://docs.ros.org/en/foxy/Tutorials/Ros2bag/Recording-And-Playing-Back-Data.html`

[4] `https://www.mathworks.com/help/matlab/ref/scatteredinterpolant.html#bvkmyb8-ExtrapolationMethod`

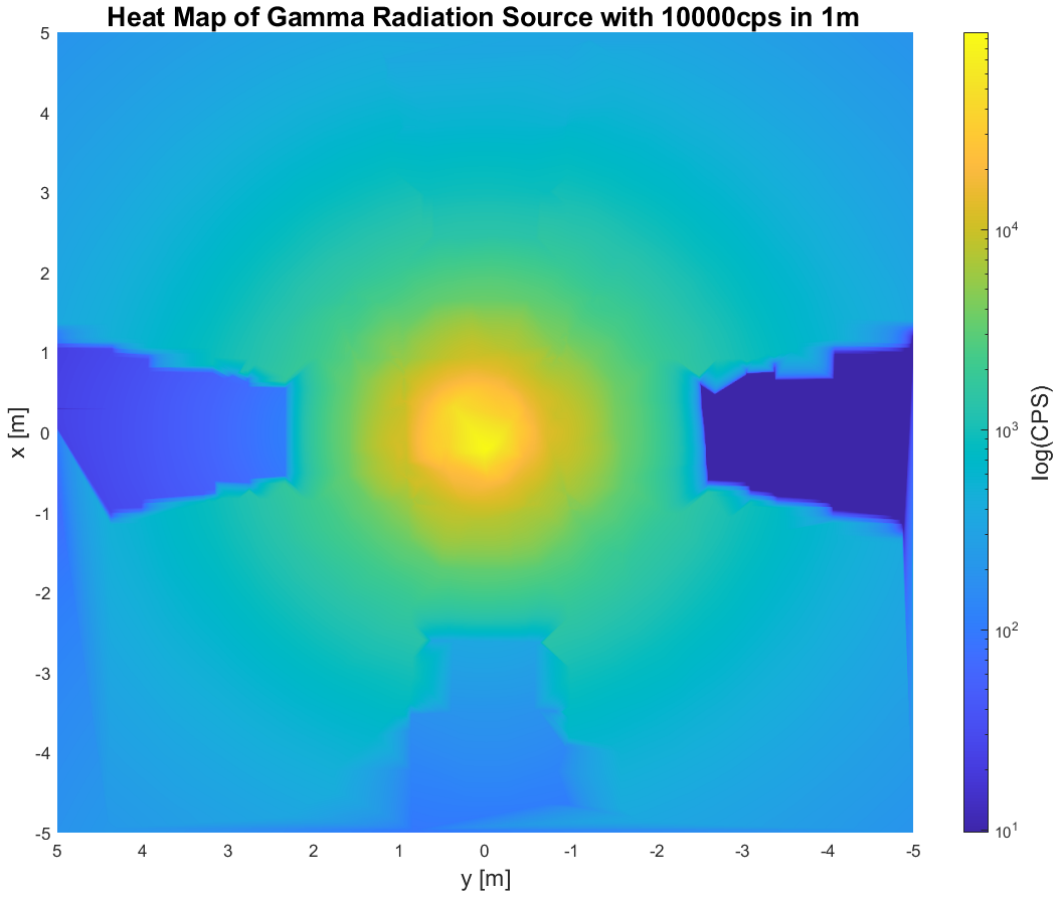


Fig. 4.3: The heat map of radiation created from data obtained from aforementioned experiment with the ideal sensor.

The knowledge of ideal radiation intensity is the next advantage of simulation. The robot can be mounted with a model of a real sensor (the next experiment), and with the hidden sensor measures the true radiation intensity.

The following graph 4.4 shows the results from the experiments mentioned above with a model of a real 2x2"NaI(Tl) scintillation detector. The sensor has a period of measurement set to $1Hz$. The robot has set a linear attenuation coefficient to a nonzero small value, and random noise was increased to 10% from the measured value. The dead time affecting real measurement is approximated with function $f(x)$[5].

$$f(x) = xe^{-0.0000197x} \tag{4.4}$$

[5]Thanks, Tomáš Lázna, for help with setting the sensor.

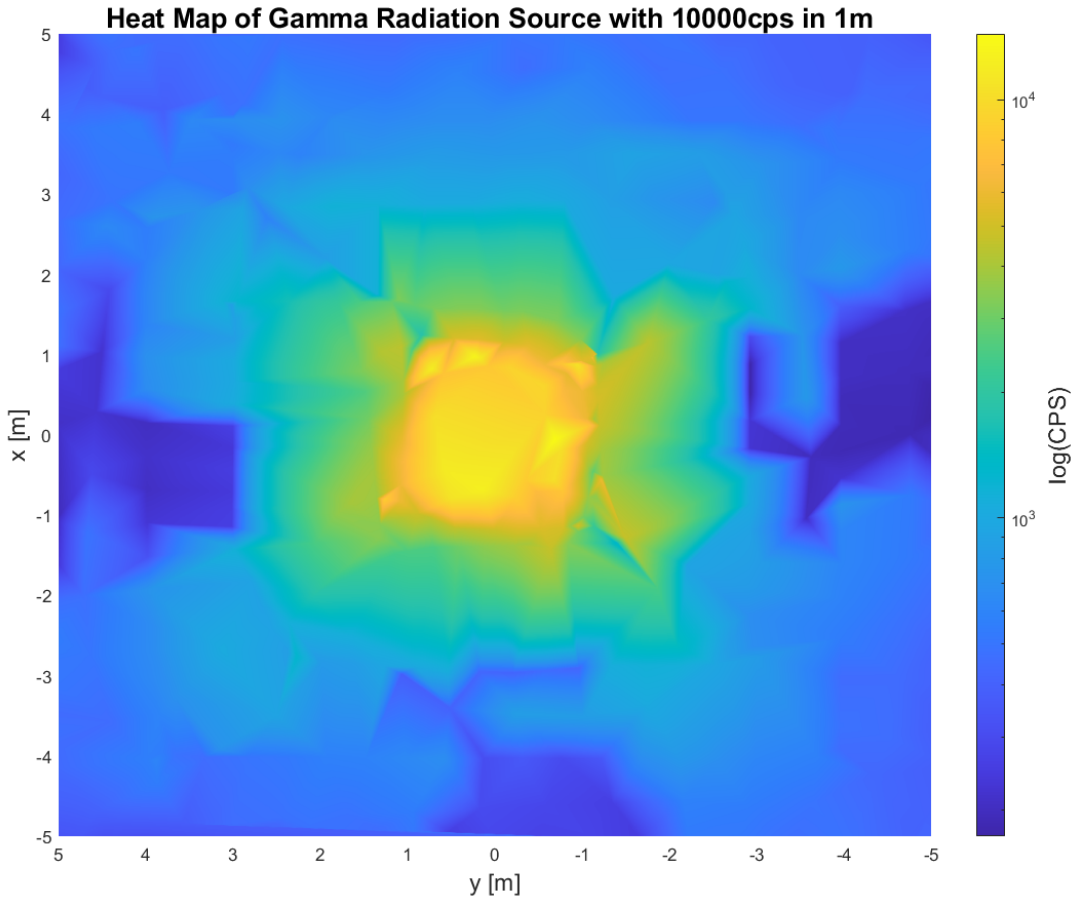


Fig. 4.4: The heat map of radiation created from data obtained from aforementioned experiment with the real sensor.

The measured value is most affected by the dead time of the sensor near the around of source, where the radiation reaches the highest values, and by the robot's orientation due to nonzero linear attenuation of the robot body. A comparison of the sensor in the simulation and the real world is shown in chapter Searching Lost Radioactive Source where the real experiment is simulated in the Gazebo simulator.

5 Searching Lost Radioactive Source

More than 120 years have passed since Henri Becquerel, Pierre Curie, and Marie Curie-Skłodowska discovered and described radioactivity. Subsequently, the first nuclear reactor was launched in 1942 in Chicago, USA. Furthermore, three years later, the first test (Trinity) of a nuclear bomb was executed. Nowadays, radioactivity is widely used in industries, medical care, national defense, agriculture, scientific research, and others. With the increasing use of radioactivity, the safe search of radiation sources becomes more necessary.

The thesis deals with two approaches to searching radiation sources. The first is based on an exhausting mapping approach when the robot's trajectory is planned before, and the trajectory covers the whole searched area. The robot measures radioactivity during the experiment. The second more active algorithm creates the robot's trajectory based on the measurement and actively searches the radiation source location.

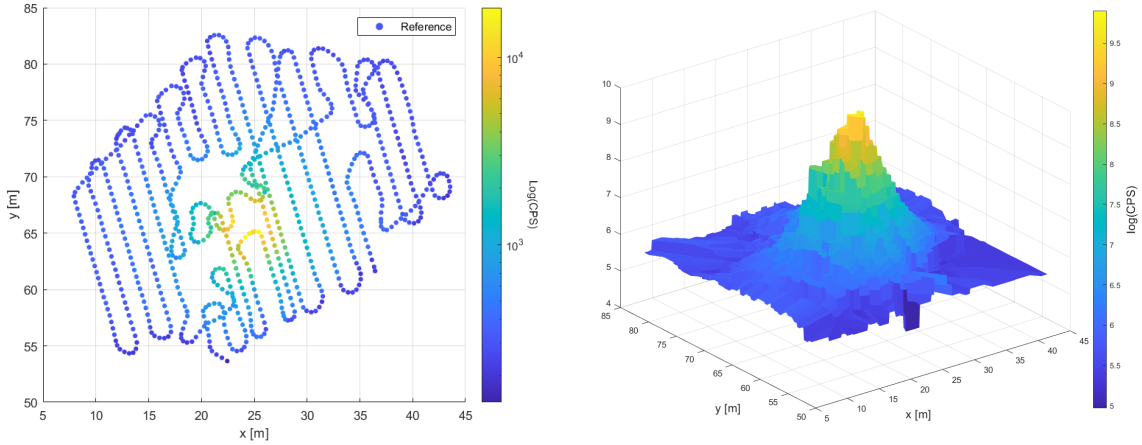
5.1 Mapping Based Approach

The mapping-based approach is a method for searching and locating the radiation source. In a known environment (map), the path is predefined and covers the region of interest. The robot measures radiation along the entire length of the trajectory, and the output is a radiation intensity map from which the position of the radiation source is calculated. An example of this trajectory is shown in following picture 5.1.

Fig. 5.1: The picture shows the terrestrial radiation mapping with a planned trajectory in a real-world experiment

The trajectory depicted in the picture 5.1 above is obtained from the Boustrophedon cell decomposition and the whole terrestrial radiation mapping experiment described in the article *An automated heterogeneous robotic system for radiation surveys: Design and field testing* [44].

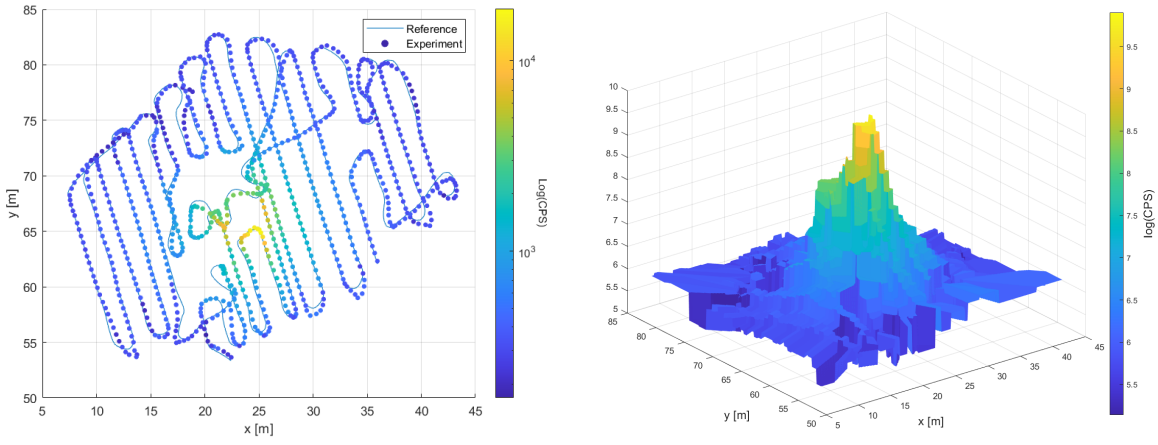
The two following pictures show the real-world measured data 5.2a and its interpolation map 5.2b.



(a) (b)

Fig. 5.2: The left picture shows the real sensor's measurement and on the right is the interpolated radiation intensity map

The maximum forward speed of the robot is $0.6ms^{-1}$ and $0.4ms^{-1}$ while turning. The experiment and measurement took 15 minutes and 10 seconds.



(a) (b)

Fig. 5.3: The simulated radiation measurement and desired trajectory are depicted on the left picture, and on the right is interpolated radiation map

The same experiments have been done with radiation sensor mode mounted on Orpheus X4 in the Gazebo simulator. The simulated experiment lasted 15 minutes and 6 seconds, and the simulated time took 21 minutes and 38 seconds with an average real-time factor 0.69.

The trajectory in the simulation slightly differs from the real trajectory because the environment model does not show all the details. Therefore, the robot cannot reach all positions like in the real-world experiment.

The source position can be obtained as the global maximum of intensity radiation map or averaging position of two peaks in the intensity map. The next methods to locate sources are mentioned in this articles [44] [65].

The presented method is convenient for detailed mapping of an area. The output intensity map provides a comprehensive view of the area in which one or more radiation sources are located. For larger areas, the mapping task becomes very time-consuming. More complex and complicated terrain can be unavailable for mobile ground robots. This case solves the use of a flying robot. The robot can discover the weak radiation sources in a large area, unlike the active search algorithm presented in the following section 5.2.

5.2 Particle Filter

A particle filter [67] [68] [69] is a group of algorithms belonging to the family of the Monte Carlo algorithms. Particle filters are used to estimate the state of the dynamic system. The goal is to estimate the radiation source position (assume 2D space; therefore, x and y coordinates are estimated), its activity, and the radiation background. The particle filter consists of five steps, usually repeated from steps **2.** to **5.**

1. **Generate a set of particles randomly**
   Each particle represents a possible solution to a state-estimation problem, and the weight indicates how much the solution matches the actual system state. The first step is to initialize each particle weight to the same value (the sum of weights is one).
2. **Predict**
   Predict the new particle state based on the real system model
3. **Update**
   Particles' states are evaluated based on the sensor measurement, and their weight is adjusted. Particles that match closely with measurement are rated higher.
4. **Resample**
   Replace the improbable particles with high probability particles (copies with slight noise) Estimate the state
5. **Compute estimation**

This section presents the specific form of a particle filter because the estimated system state is unchanging. Therefore the second step can be omitted. The rest of the section will deal with the particle filter to search lost radiation sources and estimate its parameter.

The first step is generating N particles,

$$P = (x_i, y_i, a_i, b_i, w_i)_{i=0\ldots N-1} \tag{5.1}$$

where x and y define the position, a is activity, and b denotes the background of the radiation source. Parameters x,y, and b are randomly generated with uniform distribution, and parameter a is generated with gamma distribution $\Gamma(\alpha = 2, \beta = 4000)$. Weight w_i is initialized as $1/N$.

After the measure $\vec{z} = (cps, x_s, y_s)$ is available, for each particle P_i is the computed theoretical value λ_i that the sensor would measure if the source a_i was placed

at particle position $[x_i, y_i]$.

$$\lambda_i = \frac{a_i}{(x_r - x_i)^2 + (y_r - y_i)^2 + h^2} + b_i \tag{5.2}$$

The constant h is the distance of the sensor from the ground. The weight w_i is updated according to normal distribution with probability density function

$$f(cps, \mu = \lambda_i, \sigma^2 = 15^2\lambda_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{cps-\mu}{\sigma}\right)^2} \tag{5.3}$$

and

$$w_i = \frac{f(cps, \mu = \lambda_i, \sigma = 15^2\lambda_i^2)}{\sum_{i=0}^{N-1} w_i} \tag{5.4}$$

The resample step is fulfilled with a low variance resampling [66] algorithm 1.

Algorithm 1 Low Variance Resampling

Require: P

```
P_new
r = rand(0, N^-1)
c = w_0
i = 1
n = 1
while n < N do
    u = r + n · N^-1
    while u > c do
        i = i + 1
        c = c + w_i
    end while
    P_new ← P_i
    n = n + 1
end while
```

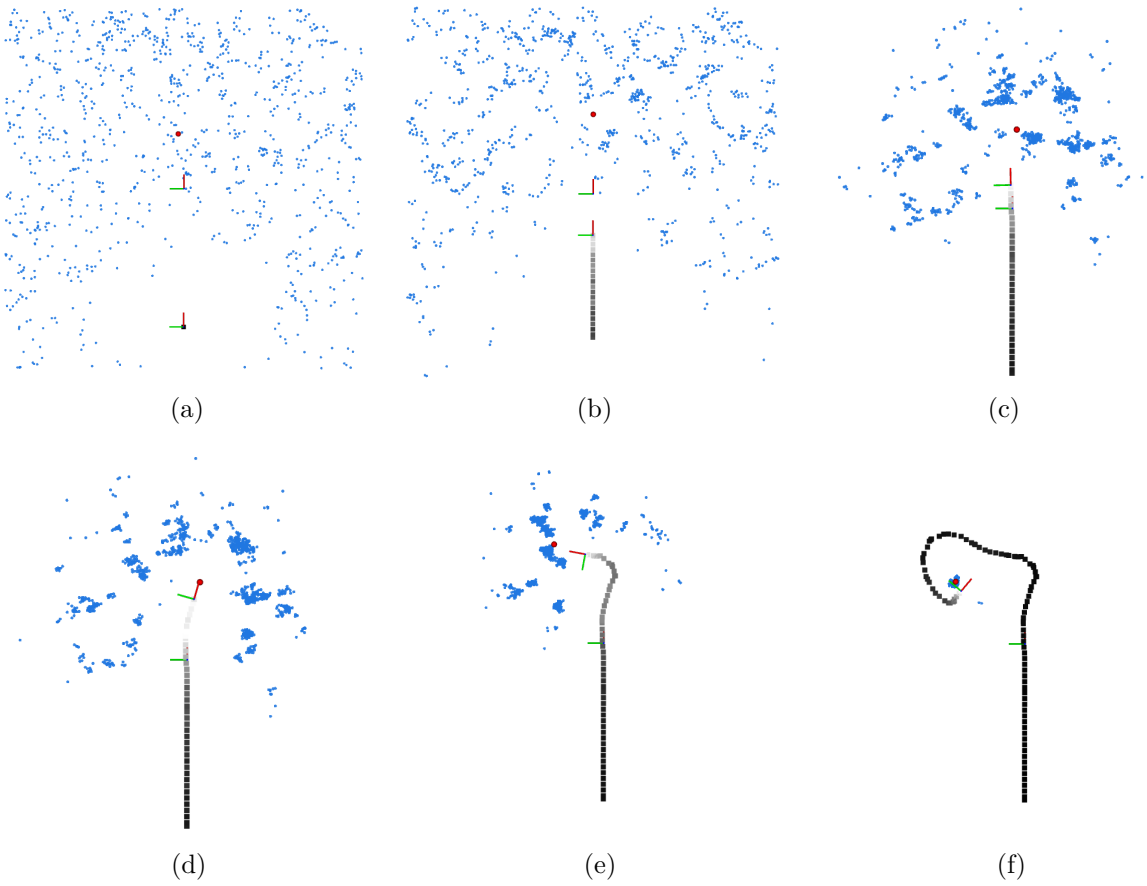
The low variance resampling algorithm picks the identical particles with higher weights repeatedly. It is inappropriate to exist the same particles, which do not bring new information to estimate the dynamic state. Therefore these particles are added to the new set of particles with small noise near the original point.

The particle filter is an iterative algorithm, and the three following experiments 5.4 5.5 5.7 are presented as six pictures from the beginning to find the goal. The experiments were performed with the Gazebo simulator, and pictures were obtained

from Rviz[1] visualization. The radiation source was placed in a squared area of $400m^2$. The summary of these experiments is shown in the table 5.1.

Tab. 5.1: Particle filter experiment overview

Order	1		2		3	
	x	y	x	y	x	y
Source Position [m]	3	3	-3	-8	-1	-2
Estimation Position [m]	3.07	3.42	-3.86	-7.19	-0.33	-0.89
Δ [m]	0.43		1.18		1.30	
CPS [-]	10000		20000		30000	
Estimate CPS	6563		7078		10673	
Background	200		200		200	
Estimate Background	196		199		205	



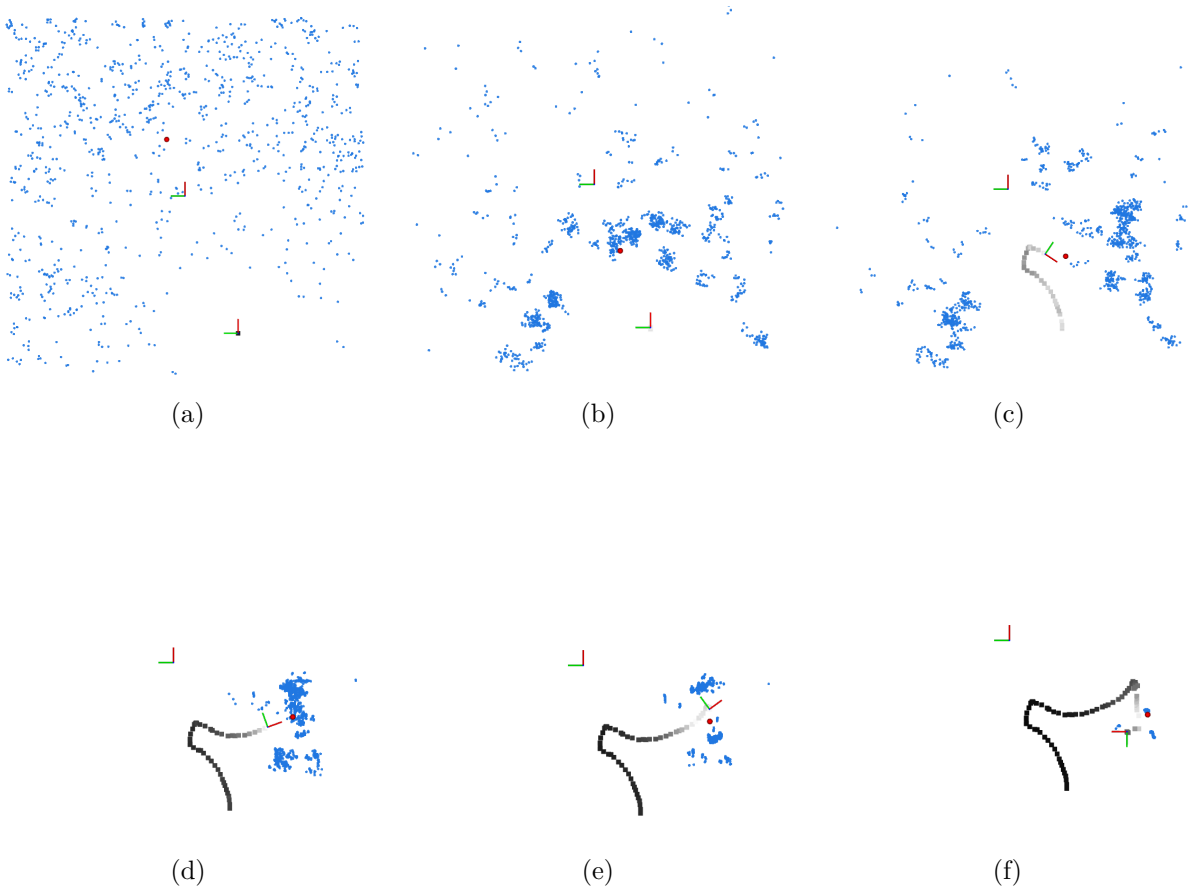
(a) (b) (c)

(d) (e) (f)

Fig. 5.4: Particle filter - first experiment

[1] `https://github.com/ros2/rviz`

The most significant advantage of the particle filter is the fast and direct radiation source search compared to the mapping-based approach. Step 5. computes the state estimate - red dot - as a weighted average from all particles. For this reason, computed estimation (in a few iterations) is placed near the center of the area. The robot always follows the red dot, i.e., the state's estimate. The fast and direct searching is balanced with lower accuracy and the possibility of non-convergence.



(a) (b) (c)

(d) (e) (f)

Fig. 5.5: Particle filter - second experiment

The images 5.5e 5.7c show cases where the particles cluster into two or three groups, and the weighted average computes a possible solution between the clusters. The non-convergence becomes 5.6 when the robot moves closer to an estimated solution, and none of the clusters outweighs.

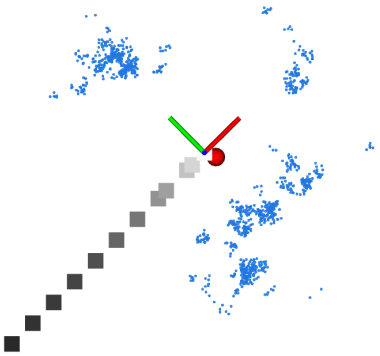
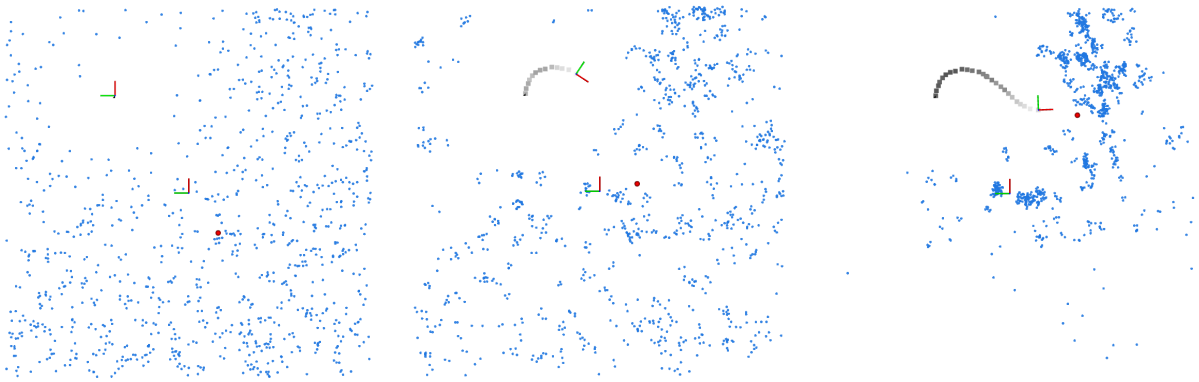


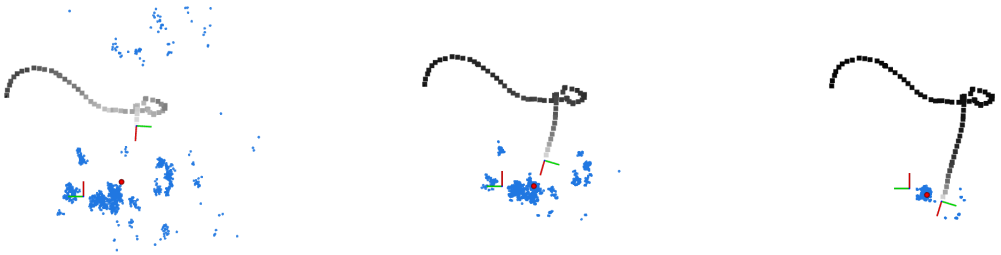
Fig. 5.6: The example of non-convergence of particle filter

The particle filter ends when the particles have the desired standard deviation σ_d.

$$\sqrt{\sigma_x^2 + \sigma_y^2} \leq \sigma_d \tag{5.5}$$



(a) (b) (c)



(d) (e) (f)

Fig. 5.7: Particle filter - third experiment

The table 5.1 shows the results of experiments with particle filter. The algorithm estimates the radiation source's position with a precision of around one meter, and the radiation background is estimated very well, with an error of up to 5%. The biggest problem is estimating the source activity, respectively, its CPS in one meter from the source. The model to compute the λ_i does not count with the nonlinear behavior of the radiation sensor near the sensor and high theoretical values of λ_i.

Future work on the particle filter will improve the convergence and better estimation of activity.

Conclusion

The thesis aims to create a robotic system based on ROS2 and a robotic simulator, which allows simulating mobile ground robots in outdoor and indoor environments. In this system are simulated algorithms for searching a lost radiation source. To fulfill these goals of the master thesis is necessary to design robot controllers for movement by trajectory and design radioactivity simulation.

The thesis first deals with a theoretical description 1of ionizing radiation, and simulators' possibilities are mentioned and compared. For the use has been preferred the Gazebo simulator before the Ignition because of the more significant variability. But the Ignition is a successor of Gazebo and will be used more in the future. The chapter with the name Environment mentions some methods to create the environment, which one is based on creating a model from point clouds. In the same chapter, the computing power requirements depend more on the number of simulated robots than on the environment's complexity. The four-wheel skid steering mobile robot was chosen to use in the outdoor environment. The chapter 3 deals with creating the model in the Gazebo simulator, and the thesis describes and visualizes the circumstances of the most significant parameters affecting the robot's behavior. Thesis describes kinematic and dynamic analysis of the SSMR, and it is compared with the robot's behavior in the simulator. The trajectory tracking problem must be solved for successful work with the robot. The thesis suggests a cubic spline for generating trajectory with continuous first two position's derivatives in waypoints. The nonlinear model of the skid steering mobile robot was linearized, and the two methods for robot motion's controller were designed. The first is based on the simple proportional-integral (PI) controller, and the next method allows to control of a continuously linearized dynamic system. The gazebo_radiation_plugin is used for simulating ionizing radiation 4 in the Gazebo. This package is modified for use in ROS2 and for simulating multiple sources with various attenuation factors for the same obstacles. The revised package is tested for ideal and real sensors with dead time, radiation background, and noise. Two methods for searching lost radiation sources were discussed and compared 5. The map-based approach provides a detailed map of radiation intensity from the whole region of interest. But for a large area, the task becomes very time-consuming. Therefore, the particle filter was implemented, which searches radiation sources directly and quickly, but its disadvantages are less accuracy and convergence problems.

Future work will focus on developing better control, path planning, navigation, localization, and more algorithms that ensure the use of robots in real-world experiments, together with the development and simulation of flying robots.

Bibliography

[1] TAKAYA Kenta, Toshinori ASAI, Valeri KROUMOV a Florentin SMARANDACHE. Simulation environment for mobile robots testing using ROS and Gazebo. In: 2016 20th International Conference on System Theory, Control and Computing (ICSTCC) [online]. IEEE, 2016, 2016, s. 96-101 [cit. 2021-12-04]. ISBN 978-1-5090-2720-0. DOI: 10.1109/ICSTCC.2016.7790647

[2] LAVRENOV Roman a Aufar ZAKIEV. Tool for 3D Gazebo Map Construction from Arbitrary Images and Laser Scans. In: 2017 10th International Conference on Developments in eSystems Engineering (DeSE) [online]. IEEE, 2017, 2017, s. 256-261 [cit. 2021-12-27]. ISBN 978-1-5386-1721-2. DOI:10.1109/DeSE.2017.33

[3] ABBYASOV Bulat, Roman LAVRENOV, Aufar ZAKIEV, Konstantin YAKOVLEV, Mikhail SVININ a Evgeni MAGID. Automatic tool for Gazebo world construction: from a grayscale image to a 3D solid model. In: 2020 IEEE International Conference on Robotics and Automation (ICRA) [online]. IEEE, 2020, 2020, s. 7226-7232 [cit. 2021-12-27]. ISBN 978-1-7281-7395-5. DOI: 10.1109/ICRA40945.2020.9196621

[4] STIBINGER Petr, Tomas BACA a Martin SASKA. Localization of Ionizing Radiation Sources by Cooperating Micro Aerial Vehicles With Pixel Detectors in Real-Time. IEEE Robotics and Automation Letters [online]. 2020, 5(2), 3634-3641 [cit. 2022-01-02]. ISSN 2377-3766. DOI: 10.1109/LRA.2020.2978456

[5] WRIGHT Thomas, Andrew WEST, Mauro LICATA, Nick HAWES a Barry LENNOX. Simulating Ionising Radiation in Gazebo for Robotic Nuclear Inspection Challenges. Robotics [online]. 2021, 10(3) [cit. 2021-12-29]. ISSN 2218-6581. DOI: 10.3390/robotics10030086

[6] Gazebo: Robot simulation made easy. [online]. Open Source Robotics Foundation, 2014 [cit. 2021-10-17]. URL: http://gazebosim.org/

[7] Non-ionizing radiation. Wikipedia: The Free Encyclopedia [online]. [cit. 2021-10-29]. URL: https://en.wikipedia.org/wiki/Non-ionizing_radiation

[8] F. KNOLL Glenn. Radiation Detection and Measurement. Third edition. Wiley India Pvt., 2009. ISBN 8126522607.

[9] Types of ionising radiation. Nuclear safety: An information portal of the Federal government and the Länder [online]. [cit. 2021-10-30]. URL: https://www.nuklearesicherheit.de/en/science/physics/ionising-radiation/types-of-ionising-radiation/

[10] HALLIDAY David, RESNICK Robert, WALKER Jearl. Fundamentals of Physics. 2010. [cit. 2021-10-30]. John Wiley & Sons Canada.

[11] Table of nuclides. Wikipedia: The Free Enciclopedia [online]. [cit. 2021-10-30]. URL: https://en.wikipedia.org/wiki/Table_of_nuclides

[12] AUGUSTYN Adam, ed. Half-life. Britannica [online]. [cit. 2021-10-31]. URL: https://www.britannica.com/science/half-life-radioactivity

[13] DE MARCILLAC Pierre, Noël CORON, Gérard DAMBIER, Jacques LEBLANC a Jean-Pierre MOALIC. Experimental detection of α-particles from the radioactive decay of natural bismuth [online]. 2003 [cit. 2021-10-31]. DOI: https://doi.org/10.1038/nature01541

[14] KRANE, Kenneth S. MODERN PHYSICS [online]. THIRD EDITION. DEPARTMENT OF PHYSICS OREGON STATE UNIVERSITY: JOHN WILEY & SONS, 2012 [cit. 2021-10-31]. ISBN ISBN 978-1-118-06114-5.

[15] Intensity (physics). Wikipedia: The Free Encyclopedia [online]. [cit. 2021-11-13]. URL: https://en.wikipedia.org/wiki/Intensity_(physics)

[16] AHMED, Syed Naeem. Physics and Engineering of Radiation Detection [online]. Queen's University, Kingston, Ontario: ELSEVIER [cit. 2021-11-13]. ISBN ISBN–13: 978-0-12-045581-2.

[17] Obecné informace o radioaktivitě a radiační ochraně. FN MOTOL [online]. [cit. 2021-11-13]. URL: https://www.fnmotol.cz/kliniky-a-oddeleni/cast-pro-dospele/klinika-nuklearni-mediciny-a-endokrinologie-uk-2-l/oddeleni-radiologicke-fyziky/obecne-informace-o-radioaktivite-a-radiacni-ochran/

[18] Internal and External Exposure. Ministry of the Environment: Government of Japan [online]. [cit. 2021-11-17]. URL: https://www.env.go.jp/en/chemi/rhm/basic-info/1st/02-01-01.html

[19] ROS: Robot Operating System [online]. [cit. 2021-11-20]. URL: https://www.ros.org/

[20] ROS wiki: Documentation [online]. [cit. 2021-11-20]. URL: https://www.ros.org/

[21] QUIGLEY Morgan, Brian GERKEY a William SMART. Programming Robots with ROS: A PRACTICAL INTRODUCTION TO THE ROBOT OPERATING SYSTEM [online]. Sebastopol, California: O'Reilly Media, 2015 [cit. 2021-11-20]. ISBN 978-1-449-32389-9.

[22] BOREN Jonathan a Steve COUSINS. Exponential Growth of ROS [ROS Topics]. IEEE Robotics & Automation Magazine [online]. 2011, 18(1), 19-20 [cit. 2021-11-20]. ISSN 1070-9932. DOI: 10.1109/MRA.2010.940147

[23] Willow Garage. Wikipedia: The free Encyclopedia [online]. [cit. 2021-11-20]. URL: https://en.wikipedia.org/wiki/Willow_Garage

[24] Open Robotics [online]. [cit. 2021-11-20]. URL: https://www.openrobotics.org/

[25] GERKEY Brian. ROS 2 Design: Why ROS 2? [online]. 2015-07 [cit. 2021-11-20]. URL: https://design.ros2.org/articles/why_ros2.html

[26] DIRK Thomas. ROS 2 Design: ROS 2 middleware interface [online]. 2017-09 [cit. 2021-11-20]. URL: http://design.ros2.org/articles/ros_middleware_interface.html

[27] ROS.org: nodelet. ROS.org: Documentation [online]. [cit. 2021-11-20]. URL: http://wiki.ros.org/nodelet

[28] MARUYAMA Yuya, Shinpei KATO a Takuya AZUMI. Exploring the performance of ROS2. In: Proceedings of the 13th International Conference on Embedded Software [online]. New York, NY, USA: ACM, 2016, 2016, s. 1-10 [cit. 2021-11-20]. ISBN 9781450344852. DOI: 10.1145/2968478.2968502

[29] INGALLS, Ricki G. Introduction to simulation. In: Proceedings of the 2011 Winter Simulation Conference (WSC) [online]. IEEE, 2011, 2011, s. 1374-1388 [cit. 2021-12-04]. ISBN 978-1-4577-2109-0. DOI: 10.1109/WSC.2011.6147858

[30] SANTOS PESSOA DE MELO, Mirella, Jose GOMES DA SILVA NETO, Pedro JORGE LIMA DA SILVA, Joao Marcelo Xavier NATARIO TEIXEIRA a Veronica TEICHRIEB. Analysis and Comparison of Robotics 3D Simulators. In: 2019 21st Symposium on Virtual and Augmented Reality (SVR) [online]. IEEE, 2019, 2019, s. 242-251 [cit. 2021-12-04]. ISBN 978-1-7281-5434-3. DOI: 10.1109/SVR.2019.00049

[31] COLLINS Jack, Shelvin CHAND, Anthony VANDERKOP a David HOWARD. A Review of Physics Simulators for Robotic Applications. IEEE Access [online]. 2021, 9, 51416-51431 [cit. 2021-12-04]. ISSN 2169-3536. DOI: 10.1109/ACCESS.2021.3068769

[32] AFZAL Afsoon, Deborah S. KATZ, Claire Le GOUES a Christopher S. TIMPERLEY. A Study on the Challenges of Using Robotics

Simulators for Testing [online]. 2020-15-5 [cit. 2021-12-04]. URL: https://www.researchgate.net/publication/340683351_A_Study_on_the_Challenges_of_Using_Robotics_Simulators_for_Testing

[33] AirSim [online]. Microsoft Research [cit. 2021-12-06]. URL: https://microsoft.github.io/AirSim/

[34] COPPELIA ROBOTICS: Coppelia Sim [online]. [cit. 2021-12-06]. URL: https://www.coppeliarobotics.com/

[35] MuJoCo: Advanced physics simulation [online]. [cit. 2021-12-06]. URL: https://mujoco.org/

[36] CARLA: Open-source simulator for autonomous driving research. [online]. [cit. 2021-12-06]. URL: https://carla.org/

[37] Webots: Open Source Robot Simulator [online]. [cit. 2021-12-06]. URL: https://cyberbotics.com/

[38] SDFormat: Describe your world. [online]. Open Source Robotics Foundation [cit. 2021-12-18]. URL: http://sdformat.org/

[39] ROS.org: Urdf/Tutorials [online]. [cit. 2021-12-06]. URL: http://wiki.ros.org/urdf/Tutorials

[40] Ignition: Simulate before you build [online]. Open Robotics. [cit. 2021-12-06]. URL: https://ignitionrobotics.org/home

[41] Open Robotics: osfr/gazebo_models [online]. Open Source Robotics Foundation, Mountain View, CA [cit. 2021-10-17]. URL: https://github.com/osrf/gazebo_models

[42] Blender - a 3D modelling and rendering package [online]. Blender Foundation, 2018 [cit. 2021-12-25]. URL: http://www.blender.org

[43] Digital elevation model. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-12-27]. URL: https://en.wikipedia.org/wiki/Digital_elevation_model

[44] GABRLIK Petr, Tomas LAZNA, Tomas JILEK, Petr SLADEK a Ludek ZALUD. An automated heterogeneous robotic system for radiation surveys: Design and field testing. Journal of Field Robotics [online]. 2021, 38(5), 657-683 [cit. 2021-12-27]. ISSN 1556-4959. DOI: 10.1002/rob.22010

[45] QGIS_software: QGIS Geographic Information System [online]. QGIS Association, 2021 [cit. 2021-12-27]. URL: https://www.qgis.org

[46] CloudCompare: 3D point cloud and mesh processing software Open Source Project [online]. [cit. 2021-12-27]. URL: https://www.cloudcompare.org/

[47] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool, Sixth Eurographics Italian Chapter Conference, page 129-136, 2008, [cit. 2021-12-27]. URL: https://www.meshlab.net

[48] KAZHDAN Misha, Ming CHUANG, Szymon RUSINKIEWICZ a Hugues HOPPE. Poisson Surface Reconstruction with Envelope Constraints. Computer Graphics Forum [online]. 2020, 39(5), 173-182 [cit. 2021-12-27]. ISSN 0167-7055. DOI: 10.1111/cgf.14077

[49] Cobalt-60. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-12-29]. URL: https://en.wikipedia.org/wiki/Cobalt-60

[50] NIST: PHYSICAL MEASUREMENT LABORATORY. : X-Ray Mass Attenuation Coefficients [online]. 2004 [cit. 2021-12-29]. URL: https://www.nist.gov/pml/x-ray-mass-attenuation-coefficients

[51] MATLAB, 2019. version R2019b, Natick, Massachusetts: The MathWorks Inc.

[52] TROJNACKI, Maciej. Dynamics Model of a Four-Wheeled Mobile Robot for Control Applications – A Three-Case Study. FILEV, D., J. JABŁKOWSKI, J. KACPRZYK, et al., ed. Intelligent Systems'2014 [online]. Cham: Springer International Publishing, 2015, 2015, s. 99-116 [cit. 2022-03-27]. Advances in Intelligent Systems and Computing. ISBN 978-3-319-11309-8. DOI: 10.1007/978-3-319-11310-4_10

[53] KOZŁOWSKI, KRZYSZTOF and DARIUSZ PAZDERSKI. MODELING AND CONTROL OF A 4-WHEEL SKID-STEERING MOBILE ROBOT. International Journal of Applied Mathematics and Computer Science [online]. 2004, 2004, 14(4), 477–496 [cit. 2022-03-28]. URL: http://matwbn.icm.edu.pl/ksiazki/amc/amc14/amc1445.pdf

[54] CARACCIOLO, L., A. DE LUCA and S. IANNITTI. Trajectory tracking control of a four-wheel differentially driven mobile robot. In: Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat.

No.99CH36288C) [online]. IEEE, 1999, s. 2632-2638 [cit. 2022-03-28]. ISBN 0-7803-5180-0. DOI: 10.1109/ROBOT.1999.773994

[55] ARSLAN, Sercan and Hakan TEMELTAŞ. Robust motion control of a four wheel drive skid-steered mobile robot. 7th International Conference on Electrical and Electronics Engineering (ELECO) [online]. 2011, 394-398 [cit. 2022-03-28].

[56] PACEJKA, Hans B. Tyre and Vehicle Dynamics [online]. Delft University of Technology, 2012 [cit. 2022-03-28]. ISBN 9780080970172. URL: http://www.engineering108.com/Data/Engineering/Automobile/tyre-and-vehicle-dynamics.pdf

[57] VIRGALA, Ivan a Michal KELEMEN. Experimental Friction Identification of a DC Motor [online]. 2013 [cit. 2022-04-21]. URL: https://www.researchgate.net/publication/253241458_Experimental_Friction_Identification_of_a_DC_Motor

[58] Geographic coordinate conversion. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2022-04-26]. URL: https://en.wikipedia.org/wiki/Geographic_coordinate_conversion

[59] CHING-LONG, Shih a Lin LI-CHEN. Trajectory Planning and Tracking Control of a Differential-Drive Mobile Robot in a Picture Drawing Application [online]. 2017 [cit. 2022-05-03]. URL: https://www.mdpi.com/2218-6581/6/3/17

[60] LEENA, N a K. K SAJU. Modelling and trajectory tracking of wheeled mobile robots. International Conference on Emerging Trends in Engineering [online]. 2016 [cit. 2022-05-03]. URL: https://www.sciencedirect.com/science/article/pii/S2212017316301839

[61] MURRAY, R. M. Lecture 2 – LQR Control [online]. CALIFORNIA INSTITUTE OF TECHNOLOGY: Control and Dynamical Systems, 2016 [cit. 2022-05-07]. URL: https://www.cds.caltech.edu/~murray/courses/cds110/wi06/lqr.pdf

[62] PRASAD, Lal Bahadur, Barjeev TYAGI a Hari Om GUPTA. Optimal Control of Nonlinear Inverted Pendulum System Using PID Controller and LQR: Performance Analysis Without and With Disturbance Input. International Journal of Automation and Computing [online]. 2016 [cit. 2022-05-07]. DOI: 0.1007/s11633-014-0818-1

[63] ZHAKATAYEV, Altay, Bexultan RAKHIM, Olzhas ADIYATOV, Almaskhan BAIMYSHEV a Huseyin Atakan VAROL. Successive linearization based model predictive control of variable stiffness actuated robots. In: 2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM) [online]. IEEE, 2017, 2017, s. 1774-1779 [cit. 2022-05-07]. ISBN 978-1-5090-5998-0. DOI: 10.1109/AIM.2017.8014275

[64] SKAF, Joëlle. Solving the LQR Problem by Block Elimination [online]. [cit. 2022-05-07]. URL: `https://stanford.edu/class/ee363/notes/riccati-derivation.pdf`

[65] LAZNA, Tomas. Optimizing the localization of gamma radiation point sources using a UGV. In: 2018 ELEKTRO [online]. IEEE, 2018, 2018, s. 1-6 [cit. 2022-05-12]. ISBN 978-1-5386-4759-2. DOI: 10.1109/ELEKTRO.2018.8398368

[66] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. Probabilistic robotics. Massachusetts: MIT Press, c2006. ISBN 978-0262201629.

[67] LABBE JR, Roger R. Kalman and Bayesian Filters in Python [online]. May 23, 2020 [cit. 2022-05-10]. URL: `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`

[68] PINKAM, Nantawat, Armagan ELIBOL a Nak Young CHONG. Informative Mobile Robot Exploration for Radiation Source Localization with a Particle Filter. In: 2020 Fourth IEEE International Conference on Robotic Computing (IRC) [online]. IEEE, 2020, 2020, s. 107-112 [cit. 2022-05-10]. ISBN 978-1-7281-5237-0. DOI: 10.1109/IRC.2020.00024

[69] ELFRING, Jos, Elena TORTA a René van de MOLENGRAFT. Particle Filters: A Hands-On Tutorial. Sensors [online]. 2021 [cit. 2022-05-10]. DOI: `https://doi.org/10.3390/s21020438`